NeXus: a common data format for neutron, x-ray, and muon science

Release 3.1

http://nexusformat.org

2013-11-25 20:21:02 GMT

CONTENTS

http://www.nexusformat.org/

CHAPTER

ONE

NEXUS: USER MANUAL

1.1 Preface

With revision 3.1 of the manual, NeXus introduced a complete version of the documentation of the NeXus standard. The content from the wiki was converted, augmented (in some parts significantly), clarified, and indexed. The NeXus Definition Language (NXDL) was introduced to define base classes and application definitions. NXDL established a method to define NeXus classes according to one of three classifications:

1. *base classes* (that represent the components used to build a NeXus data file)
2. *application definitions* (used to define a minimum set of data for a specific purpose such as scientific data processing or an instrument definition)
3. *contributed definitions* (definitions and specifications that are in an incubation status before ratification by the NIAC).

Additional examples have been added to respond to inquiries from the users of the NeXus standard about implementation and usage.

Hopefully, this improved documentation, with more examples and the new NXDL, will reduce the learning barriers incurred by those new to NeXus.

1.1.1 Representation of data examples

Most of the examples of data files have been written in a format intended to show the structure of the file rather than the data content. In some cases, where it is useful, some of the data is shown. Consider this prototype example:

example of NeXus data file structure

```
    entry:NXentry
        instrument:NXinstrument
            detector:NXdetector
                data:[]
```

```
5                       @axes = "bins"
6                       @long_name = "strip detector 1-D array"
7                       @signal = 1
8                   bins:[0, 1, 2, ... 1023]
9                       @long_name = "bin index numbers"
10          sample:NXsample
11              name = "zeolite"
12          data:NXdata
13              data --> /entry/instrument/detector/data
14              bins --> /entry/instrument/detector/bins
```

Some words on the notation:

- Hierarchy is represented by indentation. Objects on the same indentation level are in the same group
- The combination `name:NXclass` denotes a NeXus group with name `name` and class `NXclass`.
- A simple name (no following class) denotes a data field. An equal sign is used to show the value, where this is important to the example.
- Sometimes, a data type is specified and possibly a set of dimensions. For example, `energy:NX_NUMBER[NE]` says *energy* is a 1-D array of numbers (either integer or floating point) of length `NE`.
- Attributes are noted as @name=”value” pairs. The `@` symbol only indicates this is an attribute and is not part of the attribute name.
- Links are shown with a text arrow `-->` indicating the source of the link (using HDF5 notation listing the sequence of *names*).

Line 1 shows that there is one group at the root level of the file named `entry`. This group is of type `NXentry` which means it conforms to the specification of the `NXentry` NeXus base class. Using the HDF5 nomenclature, we would refer to this as the `/entry` group.

Lines 2, 10, and 12: The `/entry` group contains three subgroups: `instrument`, `sample`, and `data`. These groups are of type `NXinstrument`, `NXsample`, and `NXdata`, respectively.

Line 4: The data of this example is stored in the `/entry/instrument/detector` group in the dataset called `data` (HDF5 path is `/entry/instrument/detector/data`). The indication of `data:\[]` says that `data` is an array of unspecified dimension(s).

Lines 5-7: There are three attributes of `/entry/instrument/detector/data`: `axes`, `long_name`, and `signal`.

Line 8 (reading `bins:\[0, 1, 2, ... 1023]`) shows that `bins` is a 1-D array of length presumably 1024. A small, representative selection of values are shown.

Line 9: an attribute that shows a descriptive name of `/entry/instrument/detector/bins`. This attribute might be used by a NeXus client while plotting the data.

Line 11 (reading `name = "zeolite"`) shows how a string value is represented.

Lines 13-14: The `/entry/data`) group has two datasets that are actually linked as shown. (As you will see later, the `NXdata` group is required and enables NeXus clients to easily determine what to offer for display on a default plot.)

1.1.2 Class path specification

In some places in this documentation, a path may be shown using the class types rather than names. For example:

```
/NXentry/NXinstrument/NXcrystal/wavelength
```

identifies a dataset called `wavelength` that is inside a group of type `NXcrystal` ...

As it turns out, this syntax is the syntax used in NXDL *link* specifications. This syntax is also used when the exact name of each group is either unimportant or not specified.

If default names are taken for each class, then the above class path is expressed as this equivalent HDF5 path:

```
/entry/instrument/crystal/wavelength
```

In some places in this documentation, where clarity is needed to specify both the path and class name, you may find this equivalent path:

```
/entry:NXentry/instrument:NXinstrument/crystal:NXcrystal/wavelength
```

1.2 NeXus Introduction

In recent years, a community of scientists and computer programmers working in neutron and synchrotron facilities around the world came to the conclusion that a common data format would fulfill a valuable function in the scattering community. As instrumentation becomes more complex and data visualization become more challenging, individual scientists, or even institutions, have found it difficult to keep up with new developments. A common data format makes it easier, both to exchange experimental results and to exchange ideas about how to analyze them. It promotes greater cooperation in software development and stimulates the design of more sophisticated visualization tools. Additional background information is given in the chapter titled *Brief history of the NeXus format*.

This section is designed to give a brief introduction to NeXus, the data format and tools that have been developed in response to these needs. It explains what a modern data format such as NeXus is and how to write simple programs to read and write NeXus files.

The programmers who produce intermediate files for storing analyzed data should agree on simple interchange rules.

1.2.1 What is NeXus?

The NeXus data format has four components:

A set of design principles to help people understand what is in the data files.

A set of data storage objects (*Base Class Definitions* and *Application Definitions*) to allow the development of portable analysis software.

A set of subroutines *Utilities* and *examples* to make it easy to read and write NeXus data files.

A Scientific Community to provide the scientific data, advice, and continued involvement with the NeXus standard. NeXus provides a forum for the scientific community to exchange ideas in data storage.

In addition, NeXus relies on a set of low-level file formats to actually store NeXus files on physical media. Each of these components are described in more detail in the *Physical File format* section.

The NeXus Application-Programmer Interface (NAPI), which provides the set of subroutines for reading and writing NeXus data files, is described briefly in *NAPI: The NeXus Application Programming Interface*. (Further details are provided in the *NAPI* chapter.)

The principles guiding the design and implementation of the NeXus standard are described in the *NeXus Design* chapter.

Base classes, which comprise the data storage objects used in NeXus data files, are detailed in the *Base Class Definitions* chapter.

Additionally, a brief list describing the set of NeXus Utilities available to browse, validate, translate, and visualise NeXus data files is provided in the *NeXus Utilities* chapter.

A Set of Design Principles

NeXus data files contain four types of entity: data groups, data fields, attributes, and links.

Data Groups Data groups are like folders that can contain a number of fields and/or other groups.

Data Fields Data fields can be scalar values or multidimensional arrays of a variety of sizes (1-byte, 2-byte, 4-byte, 8-byte) and types (characters, integers, floats). In HDF, fields are represented as HDF *Scientific Data Sets* (also known as SDS).

Data Attributes Extra information required to describe a particular group or field, such as the data units, can be stored as a data attribute.

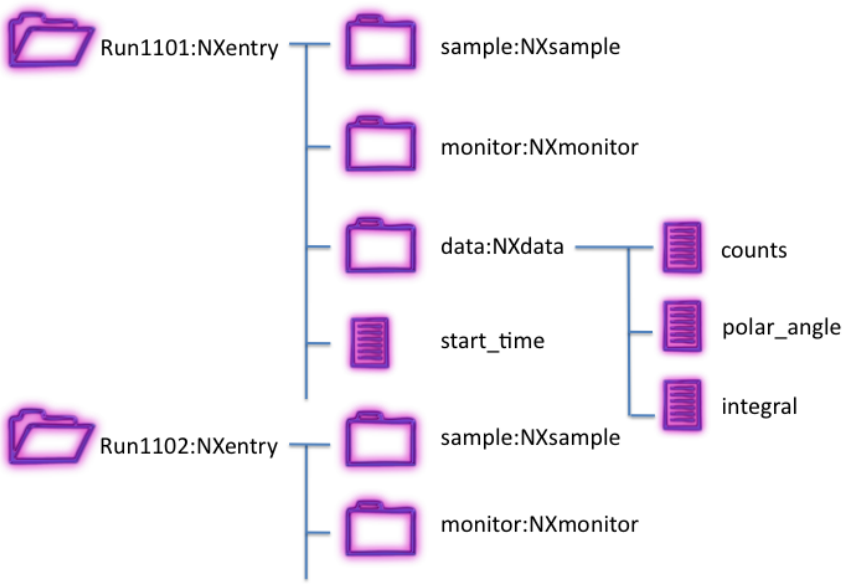
Links Links are used to reference the plottable data from `NXdata` when the data is provided in other groups such as `NXmonitor` or `NXdetector`.

In fact, a NeXus file can be viewed as a computer file system. Just as files are stored in folders (or subdirectories) to make them easy to locate, so NeXus fields are stored in groups. The group hierarchy is designed to make it easy to navigate a NeXus file.

Example of a NeXus File

The following diagram shows an example of a NeXus data file represented as a tree structure.

Example of a NeXus Data File



Note that each field is identified by a name, such as `counts`, but each group is identified both by a name and, after a colon as a delimiter, the class type, e.g., `monitor:NXmonitor`). The class types, which all begin with `NX`, define the sort of fields that the group should contain, in this case, counts from a beamline monitor. The hierarchical design, with data items nested in groups, makes it easy to identify information if you are browsing through a file.

Important Classes

Here are some of the important classes found in nearly all NeXus files. A complete list can be found in the *NeXus Design* chapter.

Note: `NXentry` and `NXdata` are the only two classes necessary to store the minimum amount of information in a valid NeXus data file.

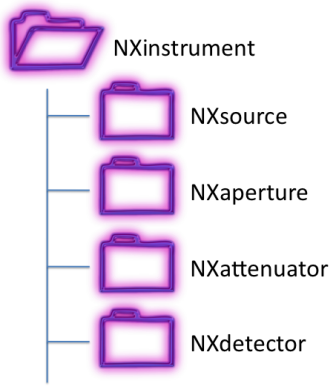
NXentry *Required:* The top level of any NeXus file contains one or more groups with the class `NXentry`. These contain all the data that is required to describe an experimental run or scan. Each `NXentry` typically contains a number of groups describing sample information (class `NXsample`), instrument details (class `NXinstrument`), and monitor counts (class `NXmonitor`).

NXdata *Required:* Each `NXentry` group contains one or more groups with class `NXdata`. These groups contain the experimental results in a self-contained way, i.e., it should be possible to generate a sensible plot of the data from the information contained in each `NXdata` group. That means it should contain the axis labels and titles as well as the data.

NXsample A `NXentry` group will often contain a group with class `NXsample`. This group contains information pertaining to the sample, such as its chemical composition, mass, and environment variables (temperature, pressure, magnetic field, etc.).

NXinstrument There might also be a group with class `NXinstrument`. This is designed to encapsulate all the instrumental information that might be relevant to a measurement, such as flight paths, collimation, chopper frequencies, etc.

`NXinstrument` excerpt



Since an instrument can include several beamline components each defined by several parameters, the components are each specified by a separate group. This hides the complexity from generic file browsers, but makes the information available in an intuitively obvious way if it is required.

Simple Example

NeXus data files do not need to be complicated. In fact, the following diagram shows an extremely simple NeXus file (in fact, the simple example shows the minimum information necessary for a NeXus data file) that could be used to transfer data between programs. (Later in this section, we show how to write and read this simple example.)

Example structure of a simple data file

This illustrates the fact that the structure of NeXus files is extremely flexible. It can accommodate very complex instrumental information, if required, but it can also be used to store very simple data sets. In the next example, a NeXus data file is shown as XML:

A very simple NeXus Data file (in XML)

```
<?xml version="1.0" encoding="UTF-8"?>
  <NXroot NeXus_version="4.3.0" XML_version="mxml"
      file_name="verysimple.xml"
      xmlns="http://definition.nexusformat.org/schema/3.1"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://definition.nexusformat.org/schema/3.1
                          http://definition.nexusformat.org/schema/3.1/BASE.xsd"
      file_time="2010-11-12T12:40:17-06:00">
    <NXentry name="entry">
      <NXdata name="data">
        <counts NAPItype="NX_INT64[15]"
                long_name="photodiode counts"
                signal="NX_INT32:1"
                axes="two_theta">
                             1193                    4474
                            53220                  274310
                           515430                  827880
                          1227100                 1434640
                          1330280                 1037070
                           598720                  316460
                            56677                    1000
                             1000
        </counts>
        <two_theta NAPItype="NX_FLOAT64[15]"
                   units="degrees"
                   long_name="two_theta (degrees)">
                  18.90940        18.90960        18.90980        18.91000
                  18.91020        18.91040        18.91060        18.91080
                  18.91100        18.91120        18.91140        18.91160
```

```
                    18.91180         18.91200         18.91220
        </two_theta>
      </NXdata>
    </NXentry>
  </NXroot>
```

NeXus files are easy to create. This example NeXus file was created using a short Python program and NeXpy:

Using NeXpy to write a very simple NeXus HDF5 Data file

```
#
# This example uses NeXpy to build the verysimple.nx5 data file.

from nexpy.api import nexus

angle = [18.9094, 18.9096, 18.9098, 18.91,  18.9102,
         18.9104, 18.9106, 18.9108, 18.911, 18.9112,
         18.9114, 18.9116, 18.9118, 18.912, 18.9122]
diode = [1193, 4474, 53220, 274310, 515430, 827880,
         1227100, 1434640, 1330280, 1037070, 598720,
         316460, 56677, 1000, 1000]

two_theta = nexus.SDS(angle, name="two_theta",
                      units="degrees",
                  long_name="two_theta (degrees)")
counts = nexus.SDS(diode, name="counts", long_name="photodiode counts")
data = nexus.NXdata(counts,[two_theta])
data.nxsave("verysimple.nx5")

# The verysimple.xml file was built with this command:
#   nxconvert -x verysimple.nx5 verysimple.xml
# and then hand-edited (line breaks) for display.
```

A Set of Data Storage Objects

If the design principles are followed, it will be easy for anyone browsing a NeXus file to understand what it contains, without any prior information. However, if you are writing specialized visualization or analysis software, you will need to know precisely what specific information is contained in advance. For that reason, NeXus provides a way of defining the format for particular instrument types, such as time-of-flight small angle neutron scattering. This requires some agreement by the relevant communities, but enables the development of much more portable software.

The set of data storage objects is divided into three parts: base classes, application definitions, and contributed definitions. The base classes represent a set of components that define the dictionary of all possible terms to be used with that component. The application definitions specify the minimum required information to satisfy a particular scientific or data analysis software interest. The contributed definitions have been submitted by the scientific community for incubation before they are adopted by the NIAC or for availability to the community.

These instrument definitions are formalized as XML files, using NXDL, (as described in the *NXDL* chapter) to specify the names of data fields, and other NeXus data objects. The following is an example of such a file for the simple NeXus file shown above.

A very simple NeXus Definition Language (NXDL) file

```
<?xml version="1.0" ?>
<definition
  xmlns="http://definition.nexusformat.org/nxdl/3.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://definition.nexusformat.org/nxdl/3.1 ../nxdl.xsd"
  category="base"
  name="verysimple"
  version="1.0"
  svnid="$Id: verysimple.nxdl.xml 1091 2012-05-28 21:10:09Z Pete Jemian $"
  type="group" extends="NXobject">

  <doc>
    A very simple NeXus NXDL file
  </doc>
  <group type="NXentry">
    <group type="NXdata">
      <field name="counts" type="NX_INT"  units="NX_UNITLESS">
        <doc>counts recorded by detector</doc>
      </field>
      <field name="two_theta" type="NX_FLOAT" units="NX_ANGLE">
        <doc>rotation angle of detector arm</doc>
      </field>
    </group>
  </group>
</definition>
```

Complete examples of reading and writing NeXus data files are provided *later*. This chapter has several examples of writing and reading NeXus data files. If you want to define the format of a particular type of NeXus file for your own use, e.g. as the standard output from a program, you are encouraged to *publish* the format using this XML format. An example of how to do this is shown in the *Creating a NXDL Specification* section.

A Set of Subroutines

NeXus data files are high-level so the user only needs to know how the data are referenced in the file but does not need to be concerned where the data are stored in the file. Thus, the data are most easily accessed using a subroutine library tuned to the specifics of the data format.

In the past, a data format was defined by a document describing the precise location of every item in the data file, either as row and column numbers in an ASCII file, or as record and byte numbers in a binary file. It is the job of the subroutine library to retrieve the data. This subroutine library is commonly called an application-programmer interface or API.

For example, in NeXus, a program to read in the wavelength of an experiment would contain lines similar to the following:

Simple example of reading data using the NeXus API

```
NXopendata (fileID, "wavelength");
NXgetdata (fileID, lambda);
NXclosedata (fileID);
```

In this example, the program requests the value of the data that has the label `wavelength`, storing the result in the variable lambda. `fileID` is a file identifier that is provided by NeXus when the file is opened.

We shall provide a more complete example when we have discussed the contents of the NeXus files.

Scientific Community

NeXus began as a group of scientists with the goal of defining a common data storage format to exchange experimental results and to exchange ideas about how to analyze them.

The *NeXus Community* provides the scientific data, advice, and continued involvement with the NeXus standard. NeXus provides a forum for the scientific community to exchange ideas in data storage through the NeXus wiki.

The NeXus International Advisory Committee supervises the development and maintenance of the NeXus common data format for neutron, X-ray, and muon science. The *NIAC: The NeXus International Advisory Committee* supervises a technical committee to oversee the *NAPI: NeXus Application Programmer Interface* and the *NeXus class definitions*.

Motivations for the NeXus standard in the Scientific Community

By the early 1990s, several groups of scientists in the fields of neutron and X-ray science had recognized a common and troublesome pattern in the data acquired at various scientific instruments and user facilities. Each of these instruments and facilities had a locally defined format for recording experimental data. With lots of different formats, much of the scientists' time was being wasted in the task of writing import readers for processing and analysis programs. As is common, the exact information to be documented from each instrument in a data file evolves, such as the implementation of new high-throughput detectors. Many of these formats lacked the generality to extend to the new data to be stored, thus another new format was devised. In such environments, the documentation of each generation of data format is often lacking.

Three parallel developments have led to NeXus:

1. *June 1994*: Mark Könnecke (Paul Scherer Institute, Switzerland) made a proposal using netCDF for the European neutron scattering community while working at the ISIS pulsed neutron facility.
2. *August 1994*: Jon Tischler and Mitch Nelson (Oak Ridge National Laboratory, USA) proposed an HDF-based format as a standard for data storage at the Advanced Photon Source (Argonne National Laboratory, USA).
3. *October 1996*: Przemek Klosowski (National Institute of Standards and Technology, USA) produced a first draft of the NeXus proposal drawing on ideas from both sources.

These scientists proposed methods to store data using a self-describing, extensible format that was already in broad use in other scientific disciplines. Their proposals formed the basis for the current design of the NeXus standard which was developed across three workshops organized by Ray Osborn (ANL), *SoftNeSS'94* (Argonne Oct. 1994), *SoftNeSS'95* (NIST Sept. 1995), and *SoftNeSS'96* (Argonne Oct. 1996), attended by representatives of a range of neutron and X-ray facilities. The NeXus API was released in late 1997. Basic motivations for this standard were:

1. *Simple plotting*
2. *Unified format for reduction and analysis*
3. *Defined dictionary of terms*

Simple plotting An important motivation for the design of NeXus was to simplify the creation of a default plot view. While the best representation of a set of observations will vary, depending on various conditions, a good suggestion is often known *a priori*. This suggestion is described in the `NXdata` element so that any program that is used to browse NeXus data files can provide a *best representation* without request for user input.

Unified format for reduction and analysis Another important motivation for NeXus, indeed the *raison d'etre*, was the community need to analyze data from different user facilities. A single data format that is in use at a variety of facilities would provide a major benefit to the scientific community. This unified format should be capable of describing any type of data from the scientific experiments, at any step of the process from data acquisition to data reduction and analysis. This unified format also needs to allow data to be written to storage as efficiently as possible to enable use with high-speed data acquisition.

Self-description, combined with a reliance on a *multi-platform* (and thereby *portable*) data storage format, are valued components of a data storage format where the longevity of the data is expected to be longer than the lifetime of the facility at which it is acquired. As the name implies, self-description within data files is the practice where the structure of the information contained within the file is evident from the file itself. A multi-platform data storage format must faithfully represent the data identically on a variety of computer systems, regardless of the bit order or byte order or word size native to the computer.

The scientific community continues to grow the various types of data to be expressed in data files. This practice is expected to continue as part of the investigative process. To gain broad acceptance in the scientific user community, any data storage format proposed as a standard would need to be *extendable* and continue to provide a means to express the latest notions of scientific data.

The maintenance cost of common data structures meeting the motivations above (self-describing, portable, and extendable) is not insurmountable but is often well-beyond the research funding of individual members of the muon, neutron, and X-ray science communities. Since it is these members that drive the selection of a data storage format, it is necessary for the user cost to be as minimal as possible. In this case, experience has shown that the format must be in the *public-domain* for it to be commonly accepted as a standard. A benefit of the public-domain aspect is that the source code for the API is open and accessible, a point which has received notable comment in the scientific literature.

More recently, NeXus has recognized that part of the scientific community with a desire to write and record scientific data, has small data volumes and a large aversion to the requirement of a complicated API necessary to access data in binary files such as HDF. For such information, the NeXus API (NAPI) has been extended by the addition of the eXtensible Markup Language (XML) [1] as an alternative to HDF. XML is a text-based format that supports compression and structured data and has broad usage in business and e-commerce. While possibly complicated, XML files are human readable, and tools for translation and extraction are plentiful. The API has routines to read and write XML data and to convert between HDF and XML.

NeXus as a Common Data Exchange Format By the late 1980s, it had become common practice for a scientific instrument or facility to define its own data format, often at the convenience of the local computer system. Data from these facilities were not easily interchanged due to various differences in computer systems and the compression schemes of binary data. It was necessary to contact the facility to obtain a description so that one could write an import routine in software. Experience with facilities closing (and subsequent lack of access to information describing the facility data format) revealed a significant limitation with this common practice. Further, there existed a *N* * *N* number of conversion routines necessary to convert data between various formats. In *N separate file formats*, circles represent different data file formats while arrows represent conversion routines. Note that the red circle only maps to one other format.

One early idea has been for NeXus to become the common data exchange format, and thereby reduce the number of data conversion routines from *N* * *N* down to *2N*, as show in *N separate file formats joined by a common NeXus converter*.

Defined dictionary of terms A necessary feature of a standard for the interchange of scientific data is a *defined dictionary* (or *lexicography*) of terms. This dictionary declares the expected spelling and meaning of terms when they are present so that it is not necessary to search for all the variant forms of *energy* when it is used to describe data (e.g., `E`, `e`, `keV`, `eV`, `nrg`, ...).

[1] XML: http://www.w3.org/XML/. There are many other descriptions of XML, for example: http://en.wikipedia.org/wiki/XML

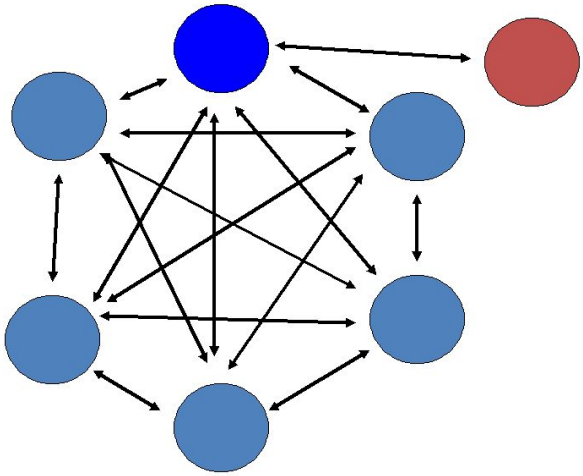


Figure 1.1: *N* separate file formats

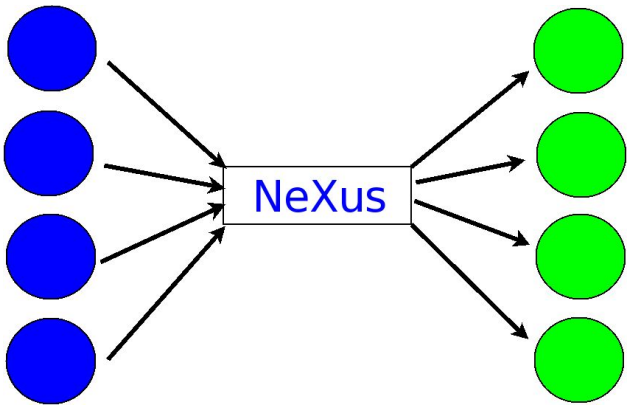


Figure 1.2: *N* separate file formats joined by a common NeXus converter

NeXus recognized that each scientific specialty has developed a unique dictionary and needs to categorize data using those terms. The NeXus Application Definitions provide the means to document the lexicography for use in data files of that scientific specialty.

NAPI: The NeXus Application Programming Interface

The NeXus API consists of routines to read and write NeXus data files. It was written to provide a simple to use and consistent common interface for all supported backends (XML, HDF4 and HDF5) to scientific programmers and other users of the NeXus Data Standard.

Note: It is not necessary to use the NAPI to write or read NeXus data files. The intent of the NAPI is to simplify the programming effort to use the HDF programming interface. There are *Examples of writing and reading NeXus data files* to help you understand.

This section will provide a brief overview of the available functionality. Further documentation of the NeXus Application Programming Interface (NAPI) for bindings to specific programming language can be found in the *NAPI* chapter and may be downloaded from the NeXus development site. [2]

For an even more detailed description of the internal workings of NAPI see `NeXusIntern.pdf`, copied from the NeXus code repository. That document is written for programmers who want to work on the NAPI itself. If you are new to NeXus and just want to implement basic file reading or writing you should not start by reading that.

How do I write a NeXus file? The NeXus Application Program Interface (NAPI) provides a set of subroutines that make it easy to read and write NeXus files. These subroutines are available in C, Fortran 77, Fortran 90, Java, Python, C++, and IDL.

The API uses a very simple *state* model to navigate through a NeXus file. (Compare this example with *NAPI Simple 2-D Write Example (C, F77, F90)*, in the *NAPI* chapter, using the native HDF5 commands.) When you open a file, the API provides a file *handle*, which stores the current location, i.e. which group and/or field is currently open. Read and write operations then act on the currently open entity. Following the simple example titled *Example structure of a simple data file*, we walk through a schematic of NeXus program written in C (without any error checking or real data).

Writing a simple NeXus file using NAPI

```
#include "napi.h"

 int main()
 {
    NXhandle fileID;
    NXopen ("NXfile.nxs", NXACC_CREATE, &fileID);
      NXmakegroup (fileID, "Scan", "NXentry");
      NXopengroup (fileID, "Scan", "NXentry");
        NXmakegroup (fileID, "data", "NXdata");
        NXopengroup (fileID, "data", "NXdata");
        /* somehow, we already have arrays tth and counts, each length n*/
          NXmakedata (fileID, "two_theta", NX_FLOAT32, 1, &n);
          NXopendata (fileID, "two_theta");
            NXputdata (fileID, tth);
            NXputattr (fileID, "units", "degrees", 7, NX_CHAR);
```

[2] http://download.nexusformat.org

```
16            NXclosedata (fileID);  /* two_theta */
17            NXmakedata (fileID, "counts", NX_FLOAT32, 1, &n);
18            NXopendata (fileID, "counts");
19              NXputdata (fileID, counts);
20            NXclosedata (fileID);  /* counts */
21          NXclosegroup (fileID);  /* data */
22        NXclosegroup (fileID);  /* Scan */
23      NXclose (&fileID);
24      return;
25  }
```

program analysis

1. **line 6:** Open the file `NXfile.nxs` with *create* access (implying write access). NAPI [3] returns a file identifier of type `NXhandle`.
2. **line 7:** Next, we create the *NXentry* group to contain the scan using `NXmakegroup()` and then open it for access using `NXopengroup()`. [4]
3. **line 9:** The plottable data is contained within an *NXdata* group, which must also be created and opened.
4. **line 12:** To create a field, call `NXmakedata()`, specifying the data name, type (`NX_FLOAT32`), rank (in this case, `1`), and length of the array (`n`). Then, it can be opened for writing. [5]
5. **line 14:** Write the data using `NXputdata()`.
6. **line 15:** With the field still open, we can also add some data attributes, such as the data units, [6] [7] which are specified as a character string (`type="NX_CHAR"` [8]) that is 7 bytes long.
7. **line 16:** Then we close the field before opening another. In fact, the API will do this automatically if you attempt to open another field, but it is better style to close it yourself.
8. **line 17:** The remaining fields in this group are added in a similar fashion. Note that the indentation whenever a new field or group are opened is just intended to make the structure of the NeXus file more transparent.
9. **line 20:** Finally, close the groups (`NXdata` and `NXentry`) before closing the file itself.

How do I read a NeXus file? Reading a NeXus file works in the same way by traversing the tree with the handle.

This schematic C code will read the two-theta array created in the *example above*. (Again, compare this example with *Reading a simple NeXus file using native HDF5 commands in C*.)

Reading a simple NeXus file using NAPI

```
1  NXopen ('NXfile.nxs', NXACC_READ, &fileID);
2    NXopengroup (fileID, "Scan", "NXentry");
3      NXopengroup (fileID, "data", "NXdata");
4        NXopendata (fileID, "two_theta");
5          NXgetinfo (fileID, &rank, dims, &datatype);
```

[3] *NAPI: NeXus Application Programmer Interface*
[4] See the chapter *Base Class Definitions* for more information.
[5] The *NeXus Data Types* section describes the available data types, such as `NX_FLOAT32` and `NX_CHAR`.
[6] *NeXus Data Units*
[7] The NeXus rule about data units is described in the *NeXus Data Units* section.
[8] see *Data Types allowed in NXDL specifications*

```
          NXmalloc ((void **) &tth, rank, dims, datatype);
          NXgetdata (fileID, tth);
        NXclosedata (fileID);
      NXclosegroup (fileID);
    NXclosegroup (fileID);
 NXclose (fileID);
```

How do I browse a NeXus file? NeXus files can also be viewed by a command-line browser, `nxbrowse`, which is included as a helper tool in the *NeXus API* distribution. The *following* is an example session of using `nxbrowse` to view a data file.

Using `nxbrowse`

```
%> nxbrowse lrcs3701.nxs

NXBrowse 3.0.0. Copyright (C) 2000 R. Osborn, M. Koennecke, P. Klosowski
    NeXus_version = 1.3.3
    file_name = lrcs3701.nxs
    file_time = 2001-02-11 00:02:35-0600
    user = EAG/RO
NX> dir
  NX Group : Histogram1 (NXentry)
  NX Group : Histogram2 (NXentry)
NX> open Histogram1
NX/Histogram1> dir
  NX Data  : title[44] (NX_CHAR)
  NX Data  : analysis[7] (NX_CHAR)
  NX Data  : start_time[24] (NX_CHAR)
  NX Data  : end_time[24] (NX_CHAR)
  NX Data  : run_number (NX_INT32)
  NX Group : sample (NXsample)
  NX Group : LRMECS (NXinstrument)
  NX Group : monitor1 (NXmonitor)
  NX Group : monitor2 (NXmonitor)
  NX Group : data (NXdata)
NX/Histogram1> read title
  title[44] (NX_CHAR) = MgB2 PDOS 43.37g 8K 120meV E0@240Hz T0@120Hz
NX/Histogram1> open data
NX/Histogram1/data> dir
  NX Data  : title[44] (NX_CHAR)
  NX Data  : data[148,750] (NX_INT32)
  NX Data  : time_of_flight[751] (NX_FLOAT32)
  NX Data  : polar_angle[148] (NX_FLOAT32)
NX/Histogram1/data> read time_of_flight
  time_of_flight[751] (NX_FLOAT32) = [ 1900.000000 1902.000000 1904.000000 ...]
    units = microseconds
    long_name = Time-of-Flight [microseconds]
NX/Histogram1/data> read data
  data[148,750] (NX_INT32) = [ 1 1 0 ...]
    units = counts
    signal = 1
    long_name = Neutron Counts
    axes = polar_angle:time_of_flight
NX/Histogram1/data> close
```

```
42 NX/Histogram1> close
43 NX> quit
```

program analysis

1. **line 1:** Start `nxbrowse` from the UNIX command line and open file `lrcs3701.nxs` from IPNS/LRMECS.
2. **line 8:** List the contents of the current group.
3. **line 11:** Open the NeXus group `Histogram1`.
4. **line 23:** Print the contents of the NeXus data labeled `title`.
5. **line 41:** Close the current group.
6. **line 43:** Quits `nxbrowse`.

The source code of `nxbrowse` [9] provides an example of how to write a NeXus reader. The test programs included in the *NeXus API* may also be useful to study.

1.3 NeXus Design

This chapter actually defines the rules to use for writing valid NeXus files. An explanation of NeXus objects is followed by the definition of NeXus coordinate systems, the rules for structuring files and the rules for storing single items of data.

The structure of NeXus files is extremely flexible, allowing the storage both of simple data sets, such as a single data array and its axes, and also of highly complex data, such as the simulation results or an entire multi-component instrument. This flexibility is a necessity as NeXus strives to capture data from a wild variety of applications in X-ray, muSR and neutron scattering. The flexibility is achieved through a hierarchical structure, with related *fields* collected together into *groups*, making NeXus files easy to navigate, even without any documentation. NeXus files are self-describing, and should be easy to understand, at least by those familiar with the experimental technique.

Note: In this manual, we use the terms *field*, *data field*, and *data item* synonymously to be consistent with their meaning between NeXus data file instances and NXDL specification files.

1.3.1 NeXus Objects and Terms

Before discussing the design of NeXus in greater detail it is necessary to define the objects and terms used by NeXus. These are:

Data Groups Group data fields and other groups together. Groups represent levels in the NeXus hierarchy

Data Fields Multidimensional arrays and scalars representing the actual data to be stored

Data Attributes Additional metadata which can be assigned to groups or data fields

Links Elements which point to data stored in another place in the file hierarchy

NeXus Base Classes Dictionaries of names possible in the various types of NeXus groups

NeXus Application Definitions Describe the minimum content of a NeXus file for a particular usage case

In the following sections these elements of NeXus files will be defined in more detail.

[9] https://svn.nexusformat.org/code/trunk/applications/NXbrowse/NXbrowse.c

Data Groups

NeXus files consist of data groups, which contain fields and/or other groups to form a hierarchical structure. This hierarchy is designed to make it easy to navigate a NeXus file by storing related fields together. Data groups are identified both by a name, which must be unique within a particular group, and a class. There can be multiple groups with the same class but they must have different names (based on the HDF rules).

For the class names used with NeXus data groups the prefix NX is reserved. Thus all NeXus class names start with NX.

Data Fields

Data fields contain the essential information stored in a NeXus file. They can be scalar values or multidimensional arrays of a variety of sizes (1-byte, 2-byte, 4-byte, 8-byte) and types (integers, floats, characters). The fields may store both experimental results (counts, detector angles, etc), and other information associated with the experiment (start and end times, user names, etc). Data fields are identified by their names, which must be unique within the group in which they are stored. Some fields have engineering units to be specified. In some cases, such in `NXdetector/data`, a field is expected to have be an array of several dimensions.

Examples of data fields

`variable` (***NX_NUMBER***) Dimension scale defining an axis of the data.

`variable_errors` (***NX_NUMBER***) Errors (uncertainties) associated with axis `variable`.

`wavelength` (***NX_FLOAT***) wavelength of radiation, `units="NX_FLOAT"`

`chemical_formula` (***NX_CHAR***) The chemical formula specified using CIF conventions.

`name` (***NX_CHAR***) Name of user responsible for this entry.

`data` (***NX_NUMBER***) Data values from the detector, `units="NX_ANY"`

Data Attributes

Attributes are extra (meta-)information that are associated with particular fields. They are used to annotate the data, e.g. with physical units or calibration offsets, and may be scalar numbers or character strings. In addition, NeXus uses attributes to identify plottable data and their axes, etc. A description of some of the many possible attributes can be found in the next table:

Examples of data attributes

`units` (***NX_CHAR***) Data units given as character strings, must conform to the NeXus units standard. See the *NeXus Data Units* section for details.

`signal` (***NX_POSINT***) Defines which data set contains the signal to be plotted, use `signal=1` for main signal, `signal=2` for a second item to plot, and so on.

`axes` (***NX_CHAR***) axes defines the names of the dimension scales for this data set as a colon-delimited list. Note that some legacy data files may use a comma as delimiter.

For example, suppose `data` is an array with elements `data[j][i]` (C) or `data(i,j)` (Fortran), with dimension scales `time_of_flight[i]` and `polar_angle[j]`, then `data` would have an attribute `axes="polar_angle:time_of_flight"` in addition to an attribute `signal=1`.

axis (*NX_POSINT*) The original way of designating data for plotting, now superceded by the `axes` attribute. This defines the rank of the signal data for which this data set is a dimension scale in order of the fastest varying index (see a longer discussion in the section on `NXdata` structure), i.e. if the array being stored is `data`, with elements `data[j][i]` in C and `data(i,j)` in Fortran, `axis` would have the following values: ith dimension (`axis=1`), jth dimension (`axis=2`), etc.

primary (*NX_POSINT*) Defines the order of preference for dimension scales which apply to the same rank of signal data. Use `primary=1` to indicate preferred dimension scale

long_name (*NX_CHAR*) Defines title of signal data or axis label of dimension scale

calibration_status (*NX_CHAR*) Defines status of data value - set to `Nominal` or `Measured`

offset (*NX_INT*) Rank values of offsets to use for each dimension if the data is not in C storage order

stride (*NX_INT*) Rank values of steps to use when incrementing the dimension

vector (*NX_FLOAT*) 3 values describing the axis of rotation or the direction of translation

interpretation (*NX_CHAR*) Describes how to display the data. Allowed values include:

- `scaler` (0-D data)
- `spectrum` (1-D data)
- `image` (2-D data)
- `vertex` (3-D data)

Finally, NeXus files themselves have global attributes which are listed in the next table. These attributes identify the NeXus version, file creation time, etc. All attributes are identified by their names, which must be unique within each field.

Examples of global attributes

file_name (*NX_CHAR*) File name of original NeXus file to assist in identification if the external name has been changed

file_time (*ISO 8601*) Date and time of file creation

file_update_time (*ISO 8601*) Date and time of last file change at close

NeXus_version (*NX_CHAR*) Version of NeXus API used in writing the file

creator (*NX_CHAR*) Facility or program where the file originated

Links

Links are pointers to existing data somewhere else. The concept is very much like symbolic links in a unix filesystem. The NeXus definition sometimes requires to have access to the same data in different groups in the same file. For example: detector data is stored in the `NXinstrument/NXdetector` group but may be needed in `NXdata` for automatic plotting. Rather then replicating the data, NeXus uses links in such situations. See the *figure* for a more descriptive representation of the concept of linking.

NeXus also allows for links to external files. Consider the case where an instrument uses a detector with a closed-system software support provided by a commercial vendor. This system writes its images into a NeXus HDF5 file. The instrument's data acquisition system writes instrument metadata into another NeXus HDF5 file. In this case, the instrument metadata file might link to the data in the detector image file. Here is an example (from Diamond Light Source) showing an external file link in HDF5:

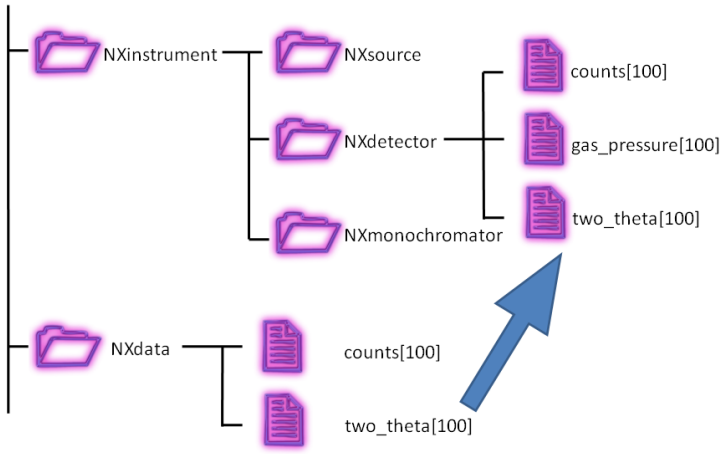


Figure 1.3: Linking in a NeXus file

Example of linking to data in an external HDF5 file

```
EXTERNAL_LINK "data" {
        TARGETFILE "/dls/i22/data/2012/sm7594-1/i22-69201-Pilatus2M.h5"
        TARGETPATH "entry/instrument/detector/data"
}
```

NeXus Base Classes

Data groups often describe objects in the experiment (monitors, detectors, monochromators, etc.), so that the contents (both data fields and/or other data groups) comprise the properties of that object. NeXus has defined a set of standard objects, or *base classes*, out of which a NeXus file can be constructed. This is each data group is identified by a name and a class. The group class, defines the type of object and the properties that it can contain, whereas the group name defines a unique instance of that class. These classes are defined in XML using the NeXus Definition Language (NXDL) format. All NeXus class types adopted by the NIAC *must* begin with `NX`. Classes not adopted by the NIAC *must not* start with `NX`.

Note: NeXus base classes are the components used to build the NeXus data structure.

Not all classes define physical objects. Some refer to logical groupings of experimental information, such as plottable data, sample environment logs, beam profiles, etc. There can be multiple instances of each class. On the other hand, a typical NeXus file will only contain a small subset of the possible classes.

Note: The groups, fields, links, and attributes of a base class definition are all **optional**, with a few particular exceptions in `NXentry` and `NXdata`. They are named in the specification to describe the exact spelling and usage of the term when it appears.

NeXus base classes are not proper classes in the same sense as used in object oriented programming languages. In fact the use of the term classes is actually misleading but has established itself during the development of NeXus. NeXus base classes are rather dictionaries of field names and their meanings which are permitted in a particular NeXus group implementing the NeXus class. This sounds complicated but becomes easy if you consider that most NeXus groups

describe instrument components. Then for example, a NXmonochromator base class describes all the possible field names which NeXus allows to be used to describe a monochromator.

Most NeXus base classes represent instrument components. Some are used as containers to structure information in a file (`NXentry`, `NXcollection`, `NXinstrument`, `NXprocess`, `NXparameter`). But there are some base classes which have special uses which need to be mentioned here:

NXdata `NXdata` is used to identify the default plottable data. The notion of a default plot of data is a basic motivation of NeXus.

NXlog `NXlog` is used to store time stamped data like the log of a temperature controller. Basically you give a start time, and arrays with a difference in seconds to the start time and the values read.

NXnote This group provides a place to store general notes, images, video or whatever. A mime type is stored together with a binary blob of data. Please use this only for auxiliary information, for example an image of your sample, or a photo of your boss.

NXgeometry `NXgeometry` and its subgroups `NXtranslation`, `NXorientation`, `NXshape` are used to store absolute positions in the laboratory coordinate system or to define shapes.

These groups can appear anywhere in the NeXus hierarchy, where needed. Preferably close to the component they annotate or in a `NXcollection`. All of the base classes are documented in the reference manual.

`NXdata` Facilitates Automatic Plotting

The most notable special base class (or *group* in NeXus) is `NXdata`. `NXdata` is the answer to a basic motivation of NeXus to facilitate automatic plotting of data. `NXdata` is designed to contain the main dataset and its associated dimension scales (axes) of a NeXus data file. The usage scenario is that an automatic data plotting program just opens a `NXentry` and then continues to search for any `NXdata` groups. These `NXdata` groups represent the plottable data. Here is the way an automatic plotting program ought to work:

1. Search for `NXentry` groups
2. Open an `NXentry`
3. Search for `NXdata` groups
4. Open an `NXdata` group
5. Identify the plottable data.
   (a) Search for a dataset with attribute `signal=1`. This is your main dataset. (There should be *only one* dataset that matches.)
   (b) Try to read the `axes` attribute of the main dataset, if it exists.
      i. The value of `axes` is a colon- or comma-separated list of the datasets describing the dimension scales (such as `axes="polar_angle:time_of_flight"`).
      ii. Parse `axes` and open the datasets to describe your dimension scales
   (c) If `axes` does not exist:
      i. Search for datasets with attributes `axis=1`, `axis=2`, etc. These are the datasets describing your axis. There may be several datasets for any axis, i.e. there may be multiple datasets with the attribute `axis=1`. Among them the dataset with the attribute `primary=1` is the preferred one. All others are alternative dimension scales.
      ii. Open the datasets to describe your dimension scales.
6. Having found the default plottable data and its dimension scales: make the plot

NeXus Application Definitions

The objects described so far provide us with the means to store data from a wide variety of instruments, simulations, or processed data as resulting from data analysis. But NeXus strives to express strict standards for certain applications of NeXus, too. The tool which NeXus uses for the expression of such strict standards is the NeXus *Application Definition*. A NeXus Application Definition describes which groups and data items have to be present in a file in order to properly describe an application of NeXus. For example for describing a powder diffraction experiment. Typically an application definition will contain only a small subset of the many groups and fields defined in NeXus. NeXus application definitions are also expressed in the NeXus Definition Language (NXDL). A tool exists which allows one to validate a NeXus file against a given application definition.

Note: NeXus application definitions define the *minimum required* information necessary to satisfy data analysis or other data processing.

Another way to look at a NeXus application definition is as a contract between a file producer (writer) and a file consumer (reader).

> The contract reads: *If you write your files following a particular NeXus application definition, I can process these files with my software.*

Yet another way to look at a NeXus application definition is to understand it as an interface definition between data files and the software which uses this file. Much like an interface in the Java or other modern object oriented programming languages.

In contrast to NeXus base classes, NeXus supports inheritance in application definitions.

Please note that a NeXus Application Definition will only define the bare minimum of data necessary to perform common analysis with data. Practical files will nearly always contain more data. One of the beauties of NeXus is that it is always possible to add more data to a file without breaking its compliance with its application definition.

1.3.2 NeXus Coordinate Systems

The NeXus coordinate system is shown *below*. Note that it is the same as that used by *McStas* (http://mcstas.risoe.dk).

Note: The NeXus definition of $+z$ is opposite to that in the IUCr International Tables for Crystallography, volume G, and consequently, $+x$ is also reversed.

Coordinate systems in NeXus have undergone significant development. Initially, only motor positions of the relevant motors were stored without further standardization. This soon proved to be too little and the *NeXus polar coordinate* system was developed. This system still is very close to angles that are meaningful to an instrument scientist but allows to define general positions of components easily. Then users from the simulation community approached the NeXus team and asked for a means to store absolute coordinates. This was implemented through the use of the *NXgeometry* class on top of the *McStas* system. We soon learned that all the things we do can be expressed through the McStas coordinate system. So it became the reference coordinate system for NeXus. `NXgeometry` was expanded to allow the description of shapes when the demand came up. Later, members of the CIF team convinced the NeXus team of the beauty of transformation matrices and NeXus was enhanced to store the necessary information to fully map CIF concepts. Not much had to be changed though as we choose to document the existing angles in CIF terms. The CIF system allows to store arbitrary operations and nevertheless calculate absolute coordinates in the laboratory coordinate system. It also allows to convert from local, for example detector coordinate systems, to absolute coordinates in the laboratory system.

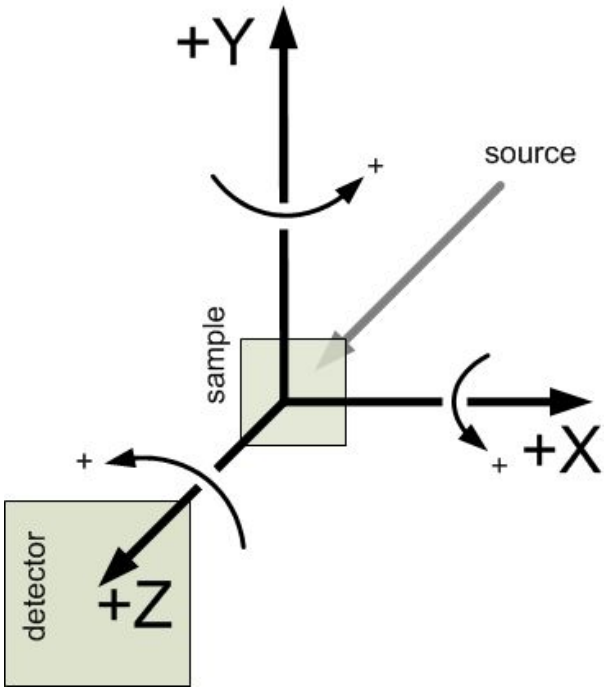


Figure 1.4: NeXus coordinate system, as viewed from detector

Please note that NXgeometry and the polar coordinate system are suggested to be deprecated. For new projects, rather use the CIF approach.

McStas and `NXgeometry` System

As stated above, NeXus uses the *McStas coordinate system* (http://mcstas.risoe.dk) as its laboratory coordinate system. The instrument is given a global, absolute coordinate system where the *z* axis points in the direction of the incident beam, the *x* axis is perpendicular to the beam in the horizontal plane pointing left as seen from the source, and the *y* axis points upwards. See below for a drawing of the McStas coordinate system. The origin of this coordinate system is the sample position or, if this is ambiguous, the center of the sample holder with all angles and translations set to zero. The McStas coordinate system is illustrated in the next figure:

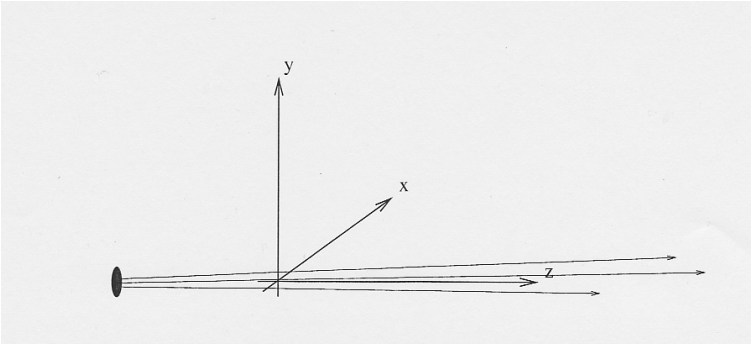


Figure 1.5: The McStas Coordinate System

The NeXus NXgeometry class directly uses the McStas coordinate system. `NXgeometry` classes can appear in any component in order to specify its position. The suggested name to use is geometry. In `NXgeometry` the

`NXtranslation/values` field defines the absolute position of the component in the McStas coordinate system. The `NXorientation/value` field describes the orientation of the component as a vector of in the McStas coordinate system.

Please note that it is planned to deprecate NXgeometry in favour of the transformation based system described below.

Simple (Spherical Polar) Coordinate System

In this system, the instrument is considered as a set of components through which the incident beam passes. The variable *distance* is assigned to each component and represents the effective beam flight path length between this component and the sample. A sign convention is used where negative numbers represent components pre-sample and positive numbers components post-sample. At each component there is local spherical coordinate system with the angles *polar_angle* and *azimuthal_angle*. The size of the sphere is the distance to the previous component.

In order to understand this spherical polar coordinate system it is helpful to look initially at the common condition that *azimuthal_angle* is zero. This corresponds to working directly in the horizontal scattering plane of the instrument. In this case *polar_angle* maps directly to the setting commonly known as *two theta*. Now, there are instruments where components live outside of the scattering plane. Most notably detectors. In order to describe such components we first apply the tilt out of the horizontal scattering plane as the *azimuthal_angle*. Then, in this tilted plane, we rotate to the component. The beauty of this is that *polar_angle* is always *two theta*. Which, in the case of a component out of the horizontal scattering plane, is not identical to the value read from the motor responsible for rotating the component. This situation is shown in *Polar Coordinate System*.

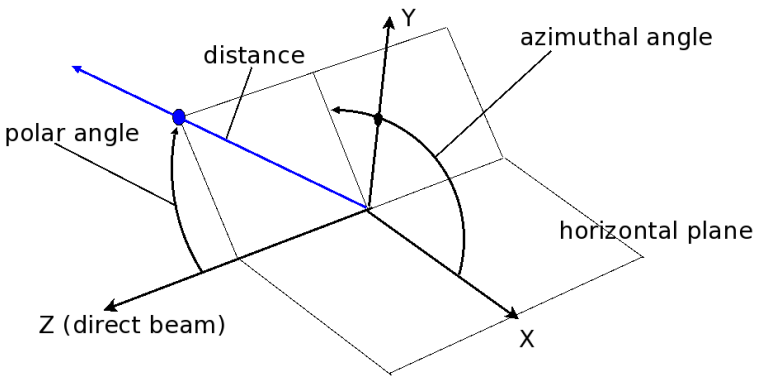


Figure 1.6: NeXus Simple (Spherical Polar) Coordinate System

Please note that it is planned to deprecate this polar system in favour of the transformation based system described below.

Coordinate Transformations

Another way to look at coordinates is through the use of transformation matrices. In this world view, the absolute position of a component or a detector pixel with respect to the laboratory coordinate system is calculated by applying a series of translations and rotations. These operations are commonly expressed as transformation matrices and their combination as matrix multiplication. A very important aspect is that the order of application of the individual operations *does* matter. Another important aspect is that any operation transforms the whole coordinate system and gives rise to a new local coordinate system. The mathematics behind this is well known and used in such applications such as industrial robot control, space flight and computer games. The beauty in this comes from the fact that the operations to apply map easily to instrument settings and constants. It is also easy to analyze the contribution of each individual operation: this can be studied under the condition that all other operations are at a zero setting.

In order to use coordinate transformations, several morsels of information need to be known:

Type The type of operation: rotation or translation

Direction The direction of the translation or the direction of the rotation axis

Value The angle of rotation or the length of the translation

Order The order of operations to apply to move a component into its place.

NeXus chooses to encode this information in the following way:

Type Through a data set attribute **transformation_type**. This can take the value of either *translation* or *rotation*.

Direction Through a data set attribute **vector**. This is a set of three values describing either the components of the rotation axis or the direction along which the translation happens.

Value This is represented in the actual data of the data set. In addition, there is the **offset** attribute which has three components describing a translation to apply before applying the operation of the real axis. Without the offset attribute additional virtual translations would need to be introduced in order to encode mechanical offsets in the axis.

Order The order is encoded through the **depends_on** attribute on a data set. The value of the **depends_on** attribute is the axis upon which the current axis sits. If the axis sits in the same group it is just a name, if it is in another group it is a path to the dependent axis. In addition, for each beamline component, there is a **depends_on** field which points to the data set at the head of the axis dependency chain. Take as an example an eulerian cradle as used on a four-circle diffractometer. Such a cradle has a dependency chain of `phi:chi:rotation_angle`. Then the `depends_on` field in *NXsample* would have the value `phi`.

NeXus Transformation encoding

Transformation encoding for an eulerian cradle on a four-circle diffractometer

```
sample:NXsample
            rotation_angle
              @transformation_type=rotation
              @vector=0,1,0
              @offset=0,0,0,
            chi
              @transformation_type=rotation
              @vector=0,0,1
              @offset=0,0,0,
              @depends_on=rotation_angle
            phi
              @transformation_type=rotation
              @vector=0,1,0
              @offset=0,0,0,
              @depends_on=chi
            depends_on
              phi
```

The type and direction of the NeXus standard operations is documented below in the table: *Actions of standard NeXus fields*. The rule is to always give the attributes to make perfectly clear how the axes work. The CIF scheme also allows to store and use arbitrarily named axes in a NeXus file.

Actions of standard NeXus fields

Transformation Actions

Field Name	transformation_type	vector
polar_angle	rotation	0 1 0
azimuthal_angle	rotation	0 0 1
meridional_angle	rotation	1 0 0
distance	translation	0 0 1
height	translation	0 1 0
x_translation	translation	1 0 0
chi	rotation	0 0 1
phi	rotation	0 1 0

For the NeXus spherical coordinate system, the order is implicit and is given in the next example.

implicit order of NeXus spherical coordinate system

```
azimuthal_angle:polar_angle:distance
```

This is also a nice example of the application of transformation matrices:

1. You first apply `azimuthal_angle` as a rotation around z. This rotates the whole coordinate out of the plane.
2. Then you apply `polar_angle` as a rotation around y in the tilted coordinate system.
3. This also moves the direction of the z vector. Along which you translate the component to place by distance.

1.3.3 Rules and Underlying File Formats

Rules for Structuring Information in NeXus Files

All NeXus files contain one or many groups of type `NXentry` at root level. Many files contain only one `NXentry` group, then the name is `entry`. The NXentry level of hierarchy is there to support the storage of multiple related experiments in one file. Or to allow the NeXus file to serve as a container for storing a whole scientific workflow from data acquisition to publication ready data. Also, `NXentry` class groups can contain raw data or processed data. For files with more than one `NXentry` group, since HDF requires that no two items at the same level in an HDF file may have the same name, the NeXus fashion is to assign names with an incrementing index appended, such as `entry1`, `entry2`, `entry3`, etc.

In order to illustrate what is written in the text, example hierarchies like the one in figure *Raw Data* are provided.

Content of a Raw Data `NXentry` Group

An example raw data hierarchy is shown in figure *Raw Data* (only showing the relevant parts of the data hierarchy). In the example shown, the `data` field in the `NXdata` group is linked to the 2-D detector data (a 512x512 array of 32-bit integers) which has the attribute `signal=1`. Note that `[,]` represents a 2D array.

NeXus Raw Data Hierarchy

```
1       entry:NXentry
2           instrument:NXinstrument
3               source:NXsource
4               ....
5               detector:NXdetector
6                   data:NX_INT32[512,512]
7                       @signal = 1
```

```
        sample:NXsample
        control:NXmonitor
        data:NXdata
            data --> /entry/instrument/detector/data
```

An NXentry describing raw data contains at least a NXsample, one NXmonitor, one NXdata and a NXinstrument group. It is good practice to use the names sample for the NXsample group, control for the NXmonitor group holding the experiment controlling monitor and instrument for the NXinstrument group. The NXinstrument group contains further groups describing the individual components of the instrument as appropriate.

The NXdata group contains links to all those data items in the NXentry hierarchy which are required to put up a default plot of the data. As an example consider a SAXS instrument with a 2D detector. The NXdata will then hold a link to the detector image. If there is only one NXdata group, it is good practice to name it data. Otherwise, the name of the detector bank represented is a good selection.

Content of a processed data NXentry group

Processed data, see figure *Processed Data*, in this context means the results of a data reduction or data analysis program. Note that [] represents a 1D array.

NeXus Processed Data Hierarchy

```
    entry:NXentry
        reduction:NXprocess
            program_name = "pyDataProc2010"
            version = "1.0a"
            input:NXparameter
                filename = "sn2013287.nxs"
        sample:NXsample
        data:NXdata
            data
                @signal = 1
```

NeXus stores such data in a simplified NXentry structure. A processed data NXentry has at minimum a NXsample, a NXdata and a NXprocess group. Again the preferred name for the NXsample group is sample. In the case of processed data, the NXdata group holds the result of the processing together with the associated axis data. The NXprocess group holds the name and version of the program used for this processing step and further NXparameter groups. These groups ought to contain the parameters used for this data processing step in suitable detail so that the processing step can be reproduced.

Optionally a processed data NXentry can hold a NXinstrument group with further groups holding relevant information about the instrument. The preferred name is again instrument. Whereas for a raw data file, NeXus strives to capture as much data as possible, a NXinstrument group for processed data may contain a much-reduced subset.

NXsubentry or Multi-Method Data

Especially at synchrotron facilities, there are experiments which perform several different methods on the sample at the same time. For example, combine a powder diffraction experiment with XAS. This may happen in the same scan, so the data needs to be grouped together. A suitable NXentry would need to adhere to two different application definitions. This leads to name clashes which cannot be easily resolved. In order to solve this issue, the following scheme was implemented in NeXus:

- The complete beamline (all data) is stored in an appropriate hierarchy in an NXentry.

- The `NXentry` group contains further `NXsubentry` groups, one for each method. Each `NXsubentry` group is constructed like a `NXentry` group. It contains links to all those data items required to fulfill the application definition for the particular method it represents.

See figure *NeXus Multi Method Hierarchy* for an example hierarchy. Note that `[,]` represents a 2D array.

NeXus Multi Method Hierarchy

```
entry:NXentry
    user:NXuser
    sample:NXsample
    instrument:NXinstument
        SASdet:NXdetector
            data:[,]
                @signal = 1
        fluordet:NXdetector
            data:[,]
                @signal = 1
        large_area:NXdetector
            data:[,]
    SAS:NXsubentry
        definition = "NXsas"
        instrument:NXinstrument
            detector:NXdetector
                data --> /entry/instrument/SASdet/data
        data:NXdata
            data --> /entry/instrument/SASdet/data
    Fluo:NXsubentry
        definition = "NXFluo"
        instrument:NXinstrument
            detector --> /entry/instrument/fluordet/data
            detector2 --> /entry/instrument/large_area/data
        data:NXdata
            detector --> /entry/instrument/fluordet/data
```

Rules for Special Cases

Scans Scans are difficult to capture because they have great variety. Basically, any variable can be scanned. Such behaviour cannot be captured in application definitions. Therefore NeXus solves this difficulty with a set of rules. In this section, `NP` is used as a symbol for the number of scan points.

- The scan dimension `NP` is always the first dimension of any multi-dimensional dataset. The reason for this is that HDF allows the first dimension of a dataset to be unlimited. Which means, that data can be appended to the dataset during the scan.
- All data is stored as arrays of dimensions `NP`, original dimensions of the data at the appropriate position in the `NXentry` hierarchy.
- The `NXdata` group has to contain links to all variables varied during the scan and the detector data. Thus the `NXdata` group mimics the usual tabular representation of a scan.
- Datasets in an `NXdata` group must contain the proper attributes to enable the default plotting, as described in the section titled *NXdata Facilitates Automatic Plotting*.

Simple scan Examples may be in order here. Let us start with a simple case, the sample is rotated around its rotation axis and data is collected in a single point detector. See figure *Simple Scan* for an overview. Then we have:

- A dataset at `NXentry/NXinstrument/NXdetector/data` of length `NP` containing the count data.
- A dataset at `NXentry/NXsample/rotation_angle` of length `NP` containing the positions of `rotation_angle` at the various steps of the scan.
- `NXdata` contains links to:
  - `NXentry/NXinstrument/NXdetector/data`
  - `NXentry/NXsample/rotation_angle`
- All other data fields have their normal dimensions.

NeXus Simple Scan Example

```
entry:NXentry
    instrument:NXinstrument
        detector:NXdetector
            data[NP]
                @signal = 1
    sample:NXsample
        rotation_angle[NP]
            @axis=1
    control:NXmonitor
        data[NP]
    data:NXdata
        data --> /entry/instrument/detector/data
        rotation_angle --> /entry/sample/rotation_angle
```

Simple scan with area detector The next example is the same scan but with an area detector with `xsize` times `ysize` pixels. The only thing which changes is that `/NXentry/NXinstrument/NXdetector/data` will have the dimensions `NP, xsize, ysize`. See figure *Simple Scan with Area Detector* for an overview.

NeXus Simple Scan Example with Area Detector

```
entry:NXentry
    instrument:NXinstrument
        detector:NXdetector
            data:[NP,xsize,ysize]
                @signal = 1
    sample:NXsample
        rotation_angle[NP]
            @axis=1
    control:NXmonitor
        data[NP]
    data:NXdata
        data --> /entry/instrument/detector/data
        rotation_angle --> /entry/sample/rotation_angle
```

Complex *hkl* scan The next example involves a complex movement along an axis in reciprocal space which requires mutiple motors of a four-circle diffractometer to be varied during the scan. We then have:

- A dataset at `NXentry/NXinstrument/NXdetector/data` of length `NP` containing the count data.

- A dataset at `NXentry/NXinstrument/NXdetector/polar_angle` of length `NP` containing the positions of the detector's polar_angle at the various steps of the scan.
- A dataset at `NXentry/NXsample/rotation_angle` of length `NP` containing the positions of `rotation_angle` at the various steps of the scan.
- A dataset at `NXentry/NXsample/chi` of length `NP` containing the positions of chi at the various steps of the scan.
- A dataset at `NXentry/NXsample/phi` of length `NP` containing the positions of phi at the various steps of the scan.
- A dataset at `NXentry/NXsample/h` of length `NP` containing the positions of the reciprocal coordinate `h` at the various steps of the scan.
- A dataset at `NXentry/NXsample/k` of length `NP` containing the positions of the reciprocal coordinate `k` at the various steps of the scan.
- A dataset at `NXentry/NXsample/l` of length `NP` containing the positions of the reciprocal coordinate `l` at the various steps of the scan.
- `NXdata` contains links to:
  - `NXentry/NXinstrument/NXdetector/data`
  - `NXentry/NXinstrument/NXdetector/polar_angle`
  - `NXentry/NXsample/rotation_angle`
  - `NXentry/NXsample/chi`
  - `NXentry/NXsample/phi`
  - `NXentry/NXsample/h`
  - `NXentry/NXsample/k`
  - `NXentry/NXsample/l`

  The datasets in `NXdata` must have the appropriate attributes as described in the axis location section.
- All other data fields have their normal dimensions.

NeXus Complex *hkl* Scan

```
entry:NXentry
    instrument:NXinstrument
        detector:NXdetector
            data[NP]
                @signal = 1
            polar_angle[NP]
                @axis = 1
            name
    sample:NXsample
        name
        rotation_angle[NP]
            @axis=1
        chi[NP]
            @axis=1
        phi[NP]
            @axis=1
        h[NP]
            @axis=1
```

```
            @primary=1
        k[NP]
            @axis=1
        l[NP]
            @axis=1
    control:NXmonitor
        data[NP]
    data:NXdata
        data --> /entry/instrument/detector/data
        rotation_angle --> /entry/sample/rotation_angle
        chi --> /entry/sample/chi
        phi --> /entry/sample/phi
        polar_angle --> /entry/instrument/detector/polar_angle
        h --> /entry/sample/h
        k --> /entry/sample/k
        l --> /entry/sample/l
```

Multi-parameter scan: XAS Data can be stored almost anywhere in the NeXus tree. While the previous examples showed data arrays in either `NXdetector` or `NXsample`, this example demonstrates that data can be stored in other places. Links are used to reference the data.

The example is for X-ray Absorption Spectroscopy (XAS) data where the monochromator energy is step-scanned and counts are read back from detectors before (`I0`) and after (`I`) the sample. These energy scans are repeated at a sequence of sample temperatures to map out, for example, a phase transition. While it is customary in XAS to plot *log(I0/I)*, we show them separately here in two different `NXdata` groups to demonstrate that such things are possible. Note that the length of the 1-D energy array is `NE` while the length of the 1-D temperature array is `NT`

NeXus Multi-parameter scan: XAS

```
entry:NXentry
    instrument:NXinstrument
        I:NXdetector
            data:NX_NUMBER[NE,NT]
                @signal = 1
                @axes = "energy:temperature"
            energy --> /entry/monochromator/energy
            temperature --> /entry/sample/temperature
        I0:NXdetector
            data:NX_NUMBER[NE,NT]
                @signal = 1
                @axes = "energy:temperature"
            energy --> /entry/monochromator/energy
            temperature --> /entry/sample/temperature
    sample:NXsample
        temperature:NX_NUMBER[NT]
    monochromator:NXmonochromator
        energy:NX_NUMBER[NE]
    I_data:NXdata
        data --> /entry/instrument/I/data
        energy --> /entry/monochromator/energy
        temperature --> /entry/sample/temperature
    I0_data:NXdata
        data --> /entry/instrument/I00/data
        energy --> /entry/monochromator/energy
        temperature --> /entry/sample/temperature
```

Rastering Rastering is the process of making experiments at various locations in the sample volume. Again, rasterisation experiments can be variable. Some people even raster on spirals! Rasterisation experiments are treated the same way as described above for scans. Just replace `NP` with `P`, the number of raster points.

Special rules apply if a rasterisation happens on a regular grid of size `xraster, yraster`. Then the variables varied in the rasterisation will be of dimensions `xraster, yraster` and the detector data of dimensions `xraster, yraster, (orginal dimensions)` of the detector. For example, an area detector of size `xsize, ysize` then it is stored with dimensions `xraster, yraster, xsize, ysize`.

Warning: Be warned: if you use the 2D rasterisation method with `xraster, yraster` you may end up with invalid data if the scan is aborted prematurely. This cannot happen if the first method is used.

NXcollection On demand from the community, NeXus introduced a more informal method of storing information in a NeXus file. This is the `NXcollection` class which can appear anywhere underneath `NXentry`. `NXcollection` is a container for holding other data. The foreseen use is to document collections of similar data which do not otherwise fit easily into the `NXinstrument` or `NXsample` hierarchy, such as the intent to record *all* motor positions on a synchrotron beamline. Thus, `NXcollection` serves as a quick point of access to data for an instrument scientist or another expert. NXcollection is also a feature for those who are t0o lazy to build up the complete NeXus hierarchy. An example usage case is documented in figure *NXcollection example*.

`NXcollection` Example

```
entry:NXentry
    positioners:NXcollection
        mxx:NXpositioner
        mzz:NXpositioner
        sgu:NXpositioner
        ttv:NXpositioner
        hugo:NXpositioner
        ....
    scalars:NXcollection
        title (dataset)
        lieselotte (dataset)
        ...
    detectors:NXcollection
        Pilatus:NXdata
        MXX-45:NXdata
        ....
```

Rules for Storing Data Items in NeXus Files

This section describes the rules which apply for storing single data fields in data files.

Naming Conventions

Group and field Names used within NeXus follow a naming convention which is made up from the following rules: The names of NeXus *group* and *field* items must only contain a restricted set of characters. This set may be described by this regular expression syntax regular expression *regular expression syntax*:

Regular expression pattern for NXDL group and field names

```
[A-Za-z_][\w_]*
```

Note that this name pattern starts with a letter (upper or lower case) or "_" (underscore), then letters, numbers, and "_" and is limited to no more than 63 characters (imposed by the HDF5 rules for names).

Sometimes it is necessary to combine words in order to build a descriptive name for a data field or a group. In such cases lowercase words are connected by underscores.

```
number_of_lenses
```

For all data fields, only names from the NeXus base class dictionaries should be used. If a data field name or even a complete component is missing, please suggest the addition to the *NIAC: The NeXus International Advisory Committee*. The addition will usually be accepted provided it is not a duplication of an existing field and adequately documented.

Note: The NeXus base classes provide a comprehensive dictionary of terms that can be used for each class. The expected spelling and definition of each term is specified in the base classes. It is not required to provide all the terms specified in a base class. Terms with other names are permitted but might not be recognized by standard software. Rather than persist in using names not specified in the standard, please suggest additions to the *NIAC: The NeXus International Advisory Committee*.

NeXus Array Storage Order

NeXus stores multi-dimensional arrays of physical values in C language storage order, where the last dimension is the fastest varying. This is the rule. *Good reasons are required to deviate from this rule.*

It is possible to store data in storage orders other than C language order.

As well it is possible to specify that the data needs to be converted first before being useful. Consider one situation, when data must be streamed to disk as fast as possible and conversion to C language storage order causes unnecessary latency. This case presents a good reason to make an exception to the standard rule.

Non C Storage Order In order to indicate that the storage order is different from C storage order two additional data set attributes, offset and stride, have to be stored which together define the storage layout of the data. Offset and stride contain rank numbers according to the rank of the multidimensional data set. Offset describes the step to make when the dimension is multiplied by 1. Stride defines the step to make when incrementing the dimension. This is best explained by some examples.

Offset and Stride for 1 D data:

```
   * raw data = 0 1 2 3 4 5 6 7 8 9
     size[1] = { 10 }  // assume uniform overall array dimensions

   * default stride:
     stride[1] = { 1 }
     offset[1] = { 0 }
     for i:
        result[i]:
           0 1 2 3 4 5 6 7 8 9

   * reverse stride:
```

```
      stride[1] = { -1 }
      offset[1] = { 9 }
      for i:
         result[i]:
            9 8 7 6 5 4 3 2 1 0
```

Offset and Stride for 2D Data

```
   * raw data = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
     size[2] = { 4, 5 }  // assume uniform overall array dimensions

   * row major (C) stride:
     stride[2] = { 5, 1 }
     offset[2] = { 0, 0 }
     for i:
        for j:
           result[i][j]:
              0 1 2 3 4
              5 6 7 8 9
              10 11 12 13 14
              15 16 17 18 19

   * column major (Fortran) stride:
     stride[2] = { 1, 4 }
     offset[2] = { 0, 0 }
     for i:
        for j:
           result[i][j]:
              0 4 8 12 16
              1 5 9 13 17
              2 6 10 14 18
              3 7 11 15 19

   * "crazy reverse" row major (C) stride:
     stride[2] = { -5, -1 }
     offset[2] = { 4, 5 }
     for i:
        for j:
           result[i][j]:
              19 18 17 16 15
              14 13 12 11 10
              9 8 7 6 5
              4 3 2 1 0
```

Offset and Stride for 3D Data

```
  * raw data = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
       20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
       40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
    size[3] = { 3, 4, 5 }  // assume uniform overall array dimensions

  * row major (C) stride:
    stride[3] = { 20, 5, 1 }
    offset[3] = { 0, 0, 0 }
    for i:
```

```
        for j:
           for k:
              result[i][j][k]:
                 0 1 2 3 4
                 5 6 7 8 9
                 10 11 12 13 14
                 15 16 17 18 19

                 20 21 22 23 24
                 25 26 27 28 29
                 30 31 32 33 34
                 35 36 37 38 39

                 40 41 42 43 44
                 45 46 47 48 49
                 50 51 52 53 54
                 55 56 57 58 59

  * column major (Fortran) stride:
     stride[3] = { 1, 3, 12 }
     offset[3] = { 0, 0, 0 }
     for i:
        for j:
           for k:
              result[i][j][k]:
                 0 12 24 36 48
                 3 15 27 39 51
                 6 18 30 42 54
                 9 21 33 45 57

                 1 13 25 37 49
                 4 16 28 40 52
                 7 19 31 43 55
                 10 22 34 46 58

                 2 14 26 38 50
                 5 17 29 41 53
                 8 20 32 44 56
                 11 23 35 47 59
```

NeXus Data Types

description	matching regular expression
integer	`NX_INT(8|16|32|64)`
floating-point	`NX_FLOAT(32|64)`
array	`(\\[0-9\\])?`
valid item name	`^[A-Za-z_][A-Za-z0-9_]*$`
valid class name	`^NX[A-Za-z0-9_]*$`

NeXus supports numeric data as either integer or floating-point numbers. A number follows that indicates the number of bits in the word. The table above shows the regular expressions that matches the data type specifier.

integers `NX_INT8`, `NX_INT16`, `NX_INT32`, or `NX_INT64`

floating-point numbers `NX_FLOAT32` or `NX_FLOAT64`

date / time stamps `NX_DATE_TIME` or `ISO8601`: Dates and times are specified using ISO-8601 standard defini-

tions. Refer to *NeXus dates and times*.

strings All strings are to be encoded in UTF-8. Since most strings in a NeXus file are restricted to a small set of characters and the first 128 characters are standard across encodings, the encoding of most of the strings in a NeXus file will be a moot point. Where encoding in UTF-8 will be important is when recording peoples names in `NXuser` and text notes in `NXnotes`. Because the few places where encoding is important also have unpredictable content, as well as the way in which current operating systems handle character encoding, it is practically impossible to test the encoding used. Hence, `nxvalidate` provides no messages relating to character encoding.

binary data Binary data is to be written as `UINT8`.

images Binary image data is to be written using `UINT8`, the same as binary data, but with an accompanying image mime-type. If the data is text, the line terminator is `[CR][LF]`.

NeXus dates and times NeXus dates and times should be stored using the ISO 8601 [10] format, e.g. `1996-07-31T21:15:22+0600`. The standard also allows for time intervals in fractional seconds with *1 or more digits of precision*. This avoids confusion, e.g. between U.S. and European conventions, and is appropriate for machine sorting.

strftime() format specifiers for ISO-8601 time

```
%Y-%m-%dT%H:%M:%S%z
```

Note: Note that the `T` appears literally in the string, to indicate the beginning of the time element, as specified in ISO 8601. It is common to use a space in place of the `T`, such as `1996-07-31 21:15:22+0600`. While human-readable (and later allowed in a relaxed revision of the standard), compatibility with libraries supporting the ISO 8601 standard is not assured with this substitution. The `strftime()` format specifier for this is "`%Y-%m-%d %H:%M:%S%z`".

NeXus Data Units

Given the plethora of possible applications of NeXus, it is difficult to define units to use. Therefore, the general rule is that you are free to store data in any unit you find fit. However, any data field must have a units attribute which describes the units, Wherever possible, SI units are preferred. NeXus units are written as a string attribute (`NX_CHAR`) and describe the engineering units. The string should be appropriate for the value. Values for the NeXus units must be specified in a format compatible with Unidata UDunits [11] Application definitions may specify units to be used for fields using an `enumeration`.

Linking Multi Dimensional Data with Axis Data

NeXus allows to store multi dimensional arrays of data. In most cases it is not sufficient to just have the indices into the array as a label for the dimensions of the data. Usually the information which physical value corresponds to an index into a dimension of the multi dimensional data set. To this purpose a means is needed to locate appropriate data arrays which describe what each dimension of a multi dimensional data set actually corresponds too. There is a standard HDF facility to do this: it is called dimension scales. Unfortunately, at a time, there was only one global namespace for dimension scales. Thus NeXus had to come up with its own scheme for locating axis data which is described here.

[10] ISO 8601: http://www.w3.org/TR/NOTE-datetime

[11] The UDunits specification also includes instructions for derived units. At present, the contents of NeXus `units` attributes are not validated in data files.

A side effect of the NeXus scheme is that it is possible to have multiple mappings of a given dimension to physical data. For example a TOF data set can have the TOF dimension as raw TOF or as energy.

There are two methods of linking each data dimension to its respective dimension scale. The preferred method uses the `axes` attribute to specify the names of each dimension scale. The original method uses the `axis` attribute to identify with an integer the axis whose value is the number of the dimension. After describing each of these methods, the two methods will be compared. A prerequisite for both methods is that the data fields describing the axis are stored together with the multi dimensional data set whose axes need to be defined in the same NeXus group. If this leads to data duplication, use links.

Linking by name using the `axes` attribute The preferred method is to define an attribute of the data itself called *axes*. The `axes` attribute contains the names of each dimension scale as a colon (or comma) separated list in the order they appear in C. For example:

Preferred way of denoting axes

```
data:NXdata
  time_of_flight = 1500.0 1502.0 1504.0 ...
  polar_angle = 15.0 15.6 16.2 ...
  some_other_angle = 0.0 0.0 2.0 ...
  data = 5 7 14 ...
    @axes = polar_angle:time_of_flight
    @signal = 1
```

Linking by dimension number using the `axis` attribute The original method is to define an attribute of each dimension scale called *axis*. It is an integer whose value is the number of the dimension, in order of fastest varying dimension. That is, if the array being stored is data with elements `data[j][i]` in C and `data(i,j)` in Fortran, where `i` is the time-of-flight index and `j` is the polar angle index, the `NXdata` group would contain:

Original way of denoting axes

```
data:NXdata
  time_of_flight = 1500.0 1502.0 1504.0 ...
    @axis = 1
    @primary = 1
  polar_angle = 15.0 15.6 16.2 ...
    @axis = 2
    @primary = 1
  some_other_angle = 0.0 0.0 2.0 ...
    @axis = 1
  data = 5 7 14 ...
    @signal = 1
```

The `axis` attribute must be defined for each dimension scale. The `primary` attribute is unique to this method of linking.

There are limited circumstances in which more than one dimension scale for the same data dimension can be included in the same `NXdata` group. The most common is when the dimension scales are the three components of an *(hkl)* scan. In order to handle this case, we have defined another attribute of type integer called `primary` whose value determines the order in which the scale is expected to be chosen for plotting, i.e.

- 1st choice: `primary=1`
- 2nd choice: `primary=2`

- etc.

If there is more than one scale with the same value of the `axis` attribute, one of them must have set `primary=1`. Defining the `primary` attribute for the other scales is optional.

Note: The `primary` attribute can only be used with the first method of defining dimension scales discussed above. In addition to the `signal` data, this group could contain a data set of the same rank and dimensions called `errors` containing the standard deviations of the data.

Discussion of the two linking methods In general the method using the `axes` attribute on the multi dimensional data set should be preferred. This leaves the actual axis describing data sets unannotated and allows them to be used as an axis for other multi dimensional data. This is especially a concern as an axis describing a data set may be linked into another group where it may describe a completely different dimension of another data set.

Only when alternative axes definitions are needed, the `axis` method should be used to specify an axis of a data set. This is shown in the example above for the `some_other_angle` field where `axis=1` denotes another possible primary axis for plotting. The default axis for plotting carries the `primary=1` attribute.

Both methods of linking data axes will be supported in NeXus utilities that identify dimension scales, such as `NXUfindaxis()`.

Storing Detectors

There are very different types of detectors out there. Storing their data can be a challenge. As a general guide line: if the detector has some well defined form, this should be reflected in the data file. A linear detector becomes a linear array, a rectangular detector becomes an array of size `xsize` times `ysize`. Some detectors are so irregular that this does not work. Then the detector data is stored as a linear array, with the index being detector number till `ndet`. Such detectors must be accompanied by further arrays of length `ndet` which give `azimuthal_angle, polar_angle and distance` for each detector.

If data from a time of flight (TOF) instrument must be described, then the TOF dimension becomes the last dimension, for example an area detector of `xsize` *vs.* `ysize` is stored with TOF as an array with dimensions `xsize, ysize, ntof`.

Monitors are Special

Monitors, detectors that measure the properties of the experimental probe rather than the sample, have a special place in NeXus files. Monitors are crucial to normalize data. To emphasize their role, monitors are not stored in the `NXinstrument` hierarchy but on `NXentry` level in their own groups as there might be multiple monitors. Of special importance is the monitor in a group called `control`. This is the main monitor against which the data has to be normalized. This group also contains the counting control information, i.e. counting mode, times, etc.

Monitor data may be multidimensional. Good examples are scan monitors where a monitor value per scan point is expected or time-of-flight monitors.

Find the plottable data

Any program whose aim is to identify plottable data should use the following procedure:

1. Open the first top level NeXus group with class `NXentry`.
2. Open the first NeXus group with class `NXdata`.

3. Loop through NeXus fields in this group searching for the item with attribute `signal="1"` indicating this field has the plottable data.
4. Check to see if this field has an attribute called `axes`. If so, the attribute value contains a colon (or comma) delimited list (in the C-order of the data array) with the names of the dimension scales associated with the plottable data. And then you can skip the next two steps.
5. If the `axes` attribute is not defined, search for the one-dimensional NeXus fields with attribute `primary=1`.
6. These are the dimension scales to label the axes of each dimension of the data.
7. Link each dimension scale to the respective data dimension by the `axis` attribute (`axis=1`, `axis=2`, ... up to the rank of the data).
8. If necessary, close the `NXdata` group, open the next one and repeat steps 3 to 6.
9. If necessary, close the `NXentry` group, open the next one and repeat steps 2 to 7.

Consult the *NeXus API* section, which describes the routines available to program these operations. In the course of time, generic NeXus browsers will provide this functionality automatically.

Physical File format

This section describes how NeXus structures are mapped to features of the underlying physical file format. This is a guide for people who wish to create NeXus files without using the NeXus-API.

Choice of HDF as Underlying File Format

At its beginnings, the founders of NeXus identified the Hierarchical Data Format (HDF) as a capable and efficient multi-platform data storage format. HDF was designed for large data sets and already had a substantial user community. HDF was developed and maintained initially by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC) and later spun off into its own group called The HDF Group (THG: http://www.hdfgroup.org/). Rather then developing its own unique physical file format, the NeXus group choose to build NeXus on top of HDF.

HDF (now HDF5) is provided with software to read and write data (this is the application-programmer interface, or API) using a large number of computing systems in common use for neutron and X-ray science. HDF is a binary data file format that supports compression and structured data.

Mapping NeXus into HDF

NeXus data structures map directly to HDF structures. NeXus *groups* are HDF4 *vgroups* or HDF5 *groups*, NeXus data sets (or *fields*) are HDF4 *SDS (scientific data sets)* or HDF5 *datasets*. Attributes map directly to HDF group or dataset attributes.

The only special case is the NeXus class name. HDF4 supports a group class which is set with the `Vsetclass()` call and read with `VGetclass()`. HDF-5 has no group class. Thus the NeXus class is stored as an attribute to the HDF-5 group with the name `NX_class` and value of the NeXus class name.

A NeXus `link` directly maps to the HDF linking mechanisms.

Note: **Examples** are provided in the *Examples of writing and reading NeXus data files* chapter. These examples include software to write and read NeXus data files using the NAPI, as well as other software examples that use native (non-NAPI) libraries. In some cases the examples show the content of the NeXus data files that are produced. Here are links to some of the examples:

- *How do I write a NeXus file?*

Perhaps the easiest way to view the implementation of NeXus in HDF5 is to view how the data structures look. For this, we use the `h5dump` command-line utility provided with the HDF5 support libraries. Short examples are provided for the basic NeXus data components:

- *group*: created in C NAPI by:

```
NXmakegroup (fileID, "entry", "NXentry");
```

- *field*: created in C NAPI by:

```
NXmakedata (fileID, "two_theta", NX_FLOAT32, 1, &n);
  NXopendata (fileID, "two_theta");
NXputdata (fileID, tth);
```

- *attribute*: created in C NAPI by:

```
NXputattr (fileID, "units", "degrees", 7, NX_CHAR);
```

- *link* created in C NAPI by:

```
# --tba--
# TODO: write some text about HDF5 hard links
# until then, see the h5dump example below
```

See the sections *NAPI Simple 2-D Write Example (C, F77, F90)* and *NAPI Python Simple 3-D Write Example* in the *Examples of writing and reading NeXus data files* chapter for examples that use the native HDF5 calls to write NeXus data files.

`h5dump` of a NeXus `NXentry` group

```
GROUP "entry" {
  ATTRIBUTE "NX_class" {
     DATATYPE  H5T_STRING {
           STRSIZE 7;
           STRPAD H5T_STR_NULLPAD;
           CSET H5T_CSET_ASCII;
           CTYPE H5T_C_S1;
        }
     DATASPACE  SCALAR
     DATA {
     (0): "NXentry"
     }
  }
  # ... group contents
}
```

h5dump of a NeXus field (HDF5 dataset)

```
 DATASET "two_theta" {
    DATATYPE  H5T_IEEE_F64LE
    DATASPACE  SIMPLE { ( 31 ) / ( 31 ) }
    DATA {
    (0): 17.9261, 17.9259, 17.9258, 17.9256, 17.9254, 17.9252,
    (6): 17.9251, 17.9249, 17.9247, 17.9246, 17.9244, 17.9243,
    (12): 17.9241, 17.9239, 17.9237, 17.9236, 17.9234, 17.9232,
    (18): 17.9231, 17.9229, 17.9228, 17.9226, 17.9224, 17.9222,
    (24): 17.9221, 17.9219, 17.9217, 17.9216, 17.9214, 17.9213,
    (30): 17.9211
    }
    ATTRIBUTE "units" {
       DATATYPE  H5T_STRING {
             STRSIZE 7;
             STRPAD H5T_STR_NULLPAD;
             CSET H5T_CSET_ASCII;
             CTYPE H5T_C_S1;
          }
       DATASPACE  SCALAR
       DATA {
       (0): "degrees"
       }
    }
    # ... other attributes
 }
```

h5dump of a NeXus attribute

```
ATTRIBUTE "axes" {
   DATATYPE  H5T_STRING {
         STRSIZE 9;
         STRPAD H5T_STR_NULLPAD;
         CSET H5T_CSET_ASCII;
         CTYPE H5T_C_S1;
      }
   DATASPACE  SCALAR
   DATA {
   (0): "two_theta"
   }
}
```

h5dump of a NeXus link

```
# NeXus links have two parts in HDF5 files.

# The dataset is created in some group.
# A "target" attribute is added to indicate the HDF5 path to this dataset.

ATTRIBUTE "target" {
   DATATYPE  H5T_STRING {
         STRSIZE 21;
         STRPAD H5T_STR_NULLPAD;
```

```
          CSET H5T_CSET_ASCII;
          CTYPE H5T_C_S1;
       }
   DATASPACE  SCALAR
   DATA {
   (0): "/entry/data/two_theta"
   }
}

# then, the hard link is created that refers to the original dataset
# (Since the name is "two_theta" in this example, it is understood that
# this link is created in a different HDF5 group than "/entry/data".)

DATASET "two_theta" {
   HARDLINK "/entry/data/two_theta"
}
```

Mapping NeXus into XML

This takes a bit more work than HDF. At the root of NeXus XML file is a XML element with the name `NXroot`. Further XML attributes to `NXroot` define the NeXus file level attributes. An example NeXus XML data file is provided in the *NeXus Introduction* chapter as Example *A very simple NeXus Data file (in XML)*.

NeXus groups are encoded into XML as elements with the name of the NeXus class and an XML attribute `name` which defines the NeXus name of the group. Further group attributes become XML attributes. An example:

NeXus group element in XML

```
    <NXentry name="entry">
    </NXentry>
```

NeXus data sets are encoded as XML elements with the name of the data. An attribute `NAPItype` defines the type and dimensions of the data. The actual data is stored as `PCDATA` [12] in the element. Another example:

NeXus data elements

```
    <mode NAPItype="NX_CHAR[7]">
        monitor
    </mode>
    <counts NAPItype="NX_INT32[4]">
        21 456  127876 319
    </counts>
```

Data are printed in appropriate formats and in C storage order. The codes understood for `NAPItype` are all the NeXus data type names. The dimensions are given in square brackets as a comma separated list. No dimensions need to be given if the data is just a single value. Data attributes are represented as XML attributes. If the attribute is not a text string, then the attribute is given in the form: *type:value*, for example: `signal="NX_POSINT:1"`.

NeXus links are stored in XML as XML elements with the name `NAPIlink` and a XML attribute `target` which stores the path to the linked entity in the file. If the item is linked under a different name, then this name is specified as a XML attribute name to the element `NAPIlink`.

[12] `PCDATA` is the XML term for *parsed character data* (see: http://www.w3schools.com/xml/xml_cdata.asp).

The authors of the NeXus API worked with the author of the miniXML XML library to create a reasonably efficient way of handling numeric data with XML. Using the NeXus API handling something like 400 detectors versus 2000 time channels in XML is not a problem. But you may hit limits with XML as the file format when data becomes to large or you try to process NeXus XML files with general XML tools. General XML tools are normally ill prepared to process large amounts of numbers.

Special Attributes

NeXus makes use of some special attributes for its internal purposes. These attributes are stored as normal group or data set attributes in the respective file format. These are:

target This attribute is automatically created when items get linked. The target attribute contains a text string with the path to the source of the item linked.

napimount The `napimount` attribute is used to implement external linking in NeXus. The string is a URL to the file and group in the external file to link too. The system is meant to be extended. But as of now, the only format supported is:

```
nxfile://path-to-file#path-infile
```

This is a NeXus file in the file system at *path-to-file* and the group *path-infile* in that NeXus file.

NAPIlink NeXus supports linking items in another group under another name. This is only supported natively in HDF-5. For HDF-4 and XML a crutch is needed. This crutch is a special class name or attribute `NAPIlink` combined with the target attribute. For groups, `NAPILink` is the group class, for data items a special attribute with the name `NAPIlink`.

1.4 Constructing NeXus Files and Application Definitions

In *NeXus Design*, we discussed the design of the NeXus format in general terms. In this section a more tutorial style introduction in how to construct a NeXus file is given. As an example a hypothetical instrument named WONI will be used.

Note: If you are looking for a tutorial on reading or writing NeXus data files using the NeXus API, consult the *NAPI: NeXus Application Programmer Interface* chapter. For code examples, refer to *Code Examples that use the NAPI* chapter. Alternatively, there are examples in the *Example NeXus C programs using native HDF5 commands* chapter of writing and reading NeXus data files using the native HDF5 interfaces in C. Further, there are also some Python examples using the `h5py` package in the *Python Examples using h5py* section.

1.4.1 The WOnderful New Instrument (WONI)

Consider yourself to be responsible for some hypothetical WOnderful New Instrument (WONI). You are tasked to ensure that WONI will record data according to the NeXus standard. For the sake of simplicity, WONI bears a strong resemblance to a simple powder diffractometer, but let's pretend that WONI cannot use any of the existing NXDL application definitions.

WONI uses collimators and a monochromator to illuminate the sample with neutrons of a selected wavelength as described in *The (fictional) WONI example powder diffractometer*. The diffracted beam is collected in a large, banana-shaped, position sensitive detector. Typical data looks like *Example Powder Diffraction Plot from (fictional) WONI at HYNES*. There is a generous background to the data plus quite a number of diffraction peaks.

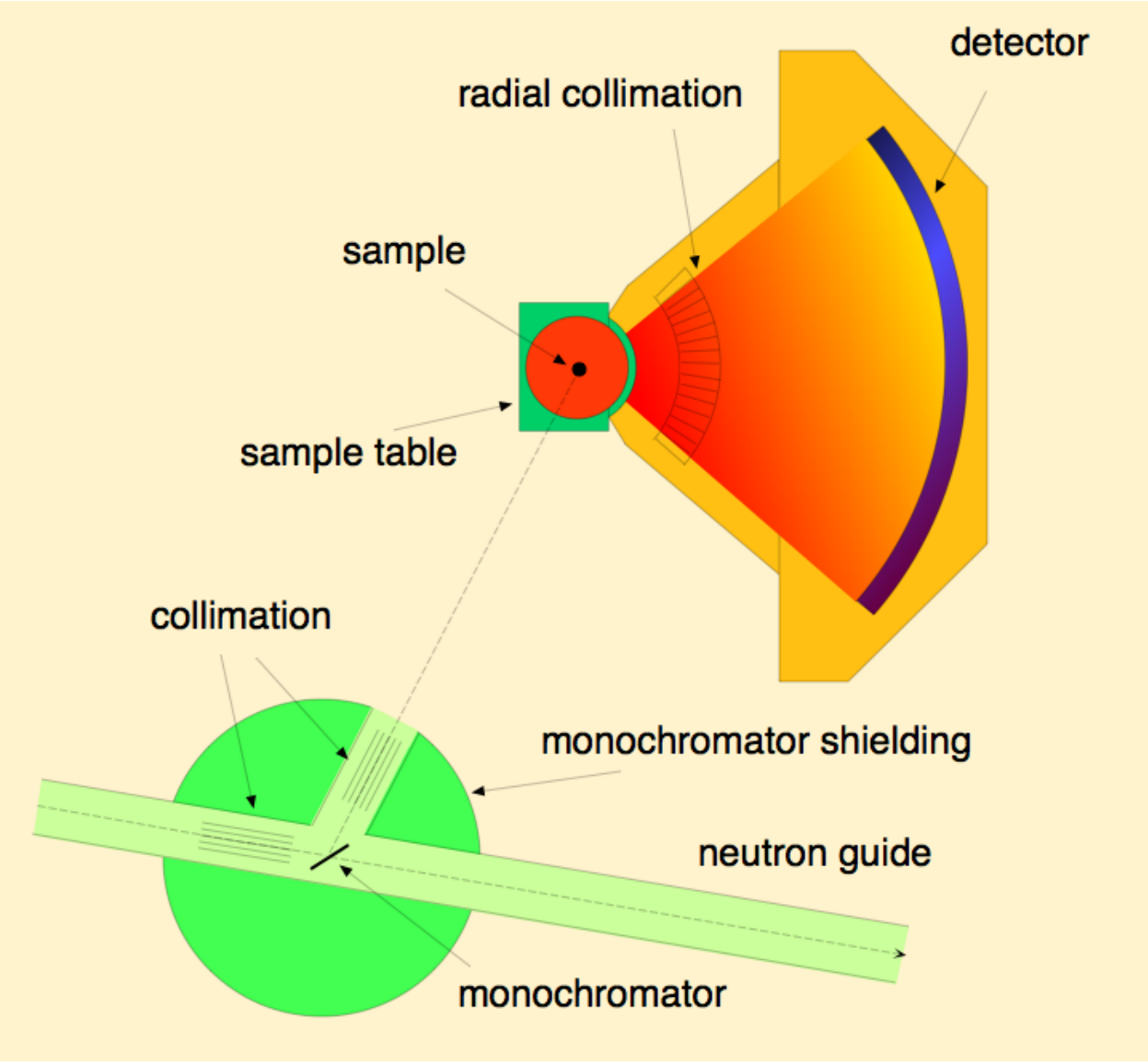


Figure 1.7: The (fictional) WONI example powder diffractometer

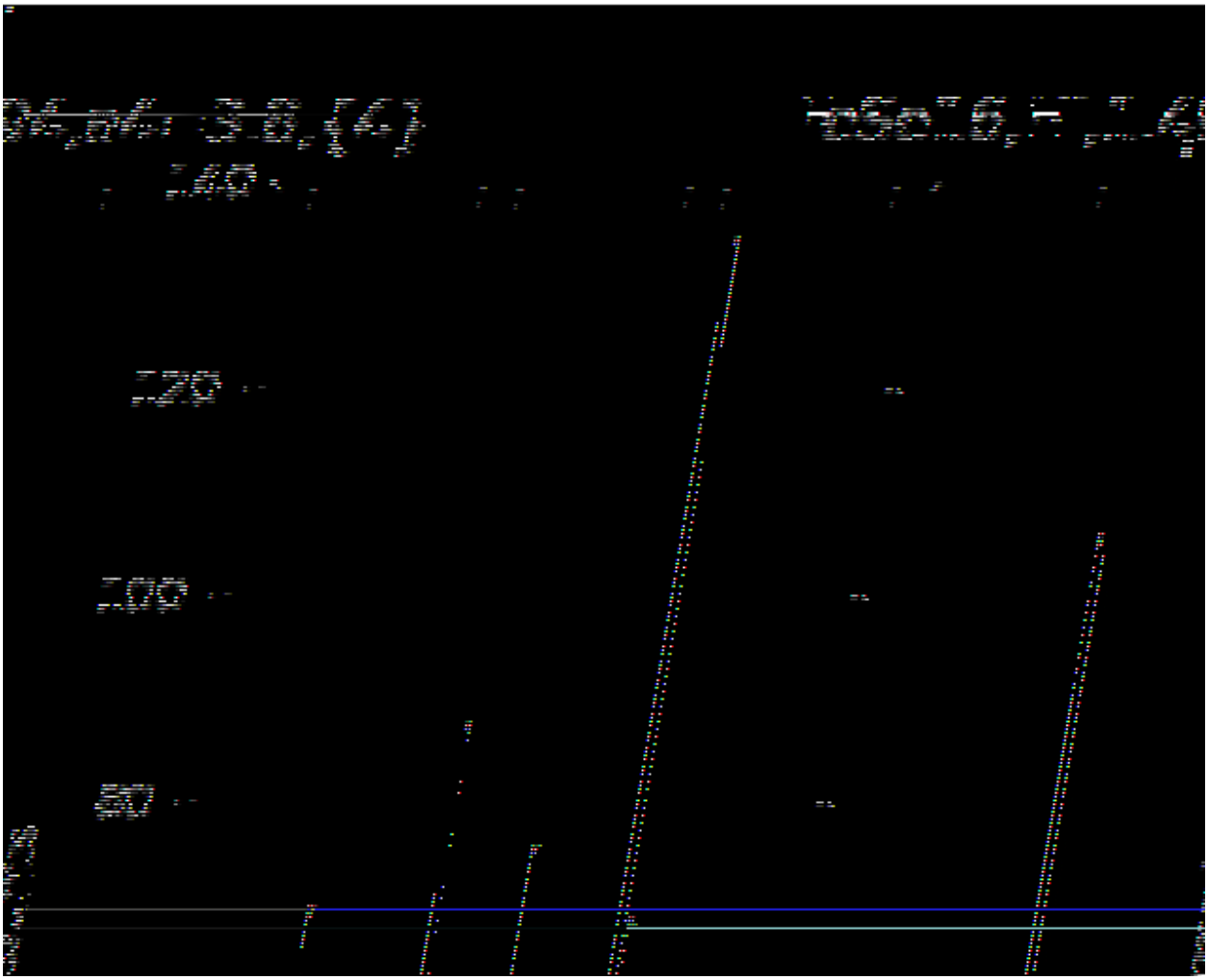


Figure 1.8: Example Powder Diffraction Plot from (fictional) WONI at HYNES

1.4.2 Constructing a NeXus file for WONI

The starting point for a NeXus file for WONI will be an empty basic NeXus file hierarchy as documented in the next figure. In order to arrive at a full NeXus file, the following steps are required:

1. For each instrument component, decide which parameters need to be stored
2. Map the component parameters to NeXus groups and parameters and add the components to the `NXinstrument` hierarchy
3. Decide what needs to go into `NXdata`
4. Fill the `NXsample` and `NXmonitor` groups

Basic structure of a NeXus file

```
entry:NXentry
  NXdata
  NXinstrument
  NXmonitor
  NXsample
```

Decide which parameters need to be stored

Now the various groups of this empty NeXus file shell need to be filled. The next step is to look at a design drawing of WONI. Identify all the instrument components like collimators, detectors, monochromators etc. For each component decide which values need to be stored. As NeXus aims to describe the experiment as good as possible, strive to capture as much information as practical.

Mapping parameters to NeXus

With the list of parameters to store for each component, consult the reference manual section on the NeXus base classes. You will find that for each of your instruments components there will be a suitable NeXus base class. Add this base class together with a name as a group under NXinstrument in your NeXus file hierarchy. Then consult the possible parameter names in the NeXus base class and match them with the parameters you wish to store for your instruments components.

As an example, consider the monochromator. You may wish to store: the wavelength, the d-value of the reflection used, the type of the monochromator and its angle towards the incoming beam. The reference manual tells you that NXcrystal is the right base class to use. Suitable fields for your parameters can be found in there to. After adding them to the basic NeXus file, the file looks like in the next figure:

Basic structure of a NeXus file with a monochromator added

```
entry:NXentry
  NXdata
  NXinstrument
     monochromator:Nxcrystal
        wavelength
        d_spacing
        rotation_angle
        reflection
        type
```

```
10      NXmonitor
11      NXsample
```

If a parameter or even a whole group is missing in order to describe your experiment, do not despair! Contact the NIAC and suggest to add the group or parameter. Give a little documentation what it is for. The NIAC will check that your suggestion is no duplicate and sufficiently documented and will then proceed to enhance the base classes with your suggestion.

A more elaborate example of the mapping process is given in the section *Creating a NXDL Specification*.

Decide on `NXdata`

The `NXdata/` group is supposed to contain the data required to put up a quick plot. For WONI this is a plot of counts versus two theta (polar_angle in NeXus) as can be seen in *Example Powder Diffraction Plot from (fictional) WONI at HYNES*. Now, in `NXdata`, create links to the appropriate data items in the `NXinstrument` hierarchy. In the case of WONI, both parameters live in the `detector:NXdetector` group.

Fill in auxiliary Information

Look at the section on `NXsample` in the NeXus reference manual. Choose appropriate parameters to store for your samples. Probably at least the name will be needed.

In order to normalize various experimental runs against each other it is necessary to know about the counting conditions and especially the monitor counts of the monitor used for normalization. The NeXus convention is to store such information in a `control:NXmonitor` group at `NXentry` level. Consult the reference for `NXmonitor` for field names. If additional monitors exist within your experiment, they will be stored as additional `NXmonitor` groups at entry level.

Consult the documentation for `NXentry` in order to find out under which names to store information such as titles, user names, experiment times etc.

A more elaborate example of this process can be found in the following section on creating an application definition.

1.4.3 Creating a NXDL Specification

An NXDL specification for a NeXus file is required if you desire to standardize NeXus files from various sources. Another name for a NXDL description is application definition. A NXDL specification can be used to verify NeXus files to conform to the standard encapsulated in the application definition. The process for constructing a NXDL specification is similar to the one described above for the construction of NeXus files.

One easy way to describe how to store data in the NeXus class structure and to create a NXDL specification is to work through an example. Along the way, we will describe some key decisions that influence our particular choices of metadata selection and data organization. So, on with the example ...

Application Definition Steps

With all this introductory stuff out of the way, let us look at the process required to define an application definition:

1. *Think!* hard about what has to go into the data file.
2. *Map* the required fields into the NeXus hierarchy
3. *Describe* this map in a NXDL file
4. *Standardize* your definition through communication with the NIAC

Step 1: *Think!* hard about data

This is actually the hard bit. There are two things to consider:

1. What has to go into the data file?
2. What is the normal plot for this type of data?

For the first part, one of the NeXus guiding principles gives us - Guidance! "A NeXus file must contain all the data necessary for standard data analysis."

Not more and not less for an application definition. Of course the definition of *standard* data for analysis or a *standard* plot depends on the science and the type of data being described. Consult senior scientists in the field about this is if you are unsure. Perhaps you must call an international meeting with domain experts to haggle that out. When considering this, people tend to put in everything which might come up. This is not the way to go.

A key test question is: Is this data item necessary for common data analysis? Only these necessary data items belong in an application definition.

The purpose of an application definition is that an author of upstream software who consumes the file can expect certain data items to be there at well defined places. On the other hand if there is a development in your field which analyzes data in a novel way and requires more data to do it, then it is better to err towards the side of more data.

Now for the case of WONI, the standard data analysis is either Rietveld refinement or profile analysis. For both purposes, the kind of radiation used to probe the sample (for WONI, neutrons), the wavelength of the radiation, the monitor (which tells us how long we counted) used to normalize the data, the counts and the two theta angle of each detector element are all required. Usually, it is desirable to know what is being analyzed, so some metadata would be nice: a title, the sample name and the sample temperature. The data typically being plotted is two theta against counts, as shown in *Example Powder Diffraction Plot from (fictional) WONI at HYNES* above. Summarizing, the basic information required from WONI is given next.

- *title* of measurement
- sample *name*
- sample *temperature*
- counts from the incident beam *monitor*
- type of radiation *probe*
- *wavelength* (λ) of radiation incident on sample
- angle (2θ or *two theta*) of detector elements
- *counts* for each detector element

If you start to worry that this is too little information, hold on, the section on Using an Application Definition (*Using an Application Definition*) will reveal the secret how to go from an application definition to a practical file.

Step 2: *Map* Data into the NeXus Hierarchy

This step is actually easier then the first one. We need to map the data items which were collected in Step 1 into the NeXus hierarchy. A NeXus file hierarchy starts with an `NXentry` group. At this stage it is advisable to pull up the base class definition for `NXentry` and study it. The first thing you might notice is that `NXentry` contains a field named `title`. Reading the documentation, you quickly realize that this is a good place to store our title. So the first mapping has been found.

```
title = /NXentry/title
```

Note: In this example, the mapping descriptions just contain the path strings into the NeXus file hierarchy with the class names of the groups to use. As it turns out, this is the syntax used in NXDL link specifications. How convenient!

Another thing to notice in the `NXentry` base class is the existence of a group of class `NXsample`. This looks like a great place to store information about the sample. Studying the `NXsample` base class confirms this view and there are two new mappings:

```
sample name = /NXentry/NXsample/name
sample temperature = /NXentry/NXsample/temperature
```

Scanning the `NXentry` base class further reveals there can be a `NXmonitor` group at this level. Looking up the base class for `NXmonitor` reveals that this is the place to store our monitor information.

```
monitor = /NXentry/NXmonitor/data
```

For the other data items, there seem to be no solutions in `NXentry`. But each of these data items describe the instrument in more detail. NeXus stores instrument descriptions in the `/NXentry/NXinstrument` branch of the hierarchy. Thus, we continue by looking at the definition of the `NXinstrument` base class. In there we find further groups for all possible instrument components. Looking at the schematic of WONI (*The (fictional) WONI example powder diffractometer*), we realize that there is a source, a monochromator and a detector. Suitable groups can be found for these components in `NXinstrument` and further inspection of the appropriate base classes reveals the following further mappings:

```
probe = /NXentry/NXinstrument/NXsource/probe
wavelength = /NXentry/NXinstrument/NXcrystal/wavelength
two theta of detector elements = /NXentry/NXinstrument/NXdetector/polar angle
counts for each detector element = /NXentry/NXinstrument/NXdetector/data
```

Thus we mapped all our data items into the NeXus hierarchy! What still needs to be done is to decide upon the content of the `NXdata` group in `NXentry`. This group describes the data necessary to make a quick plot of the data. For WONI this is `counts` versus `two theta`. Thus we add this mapping:

```
two theta of detector elements = /NXentry/NXdata/polar angle
counts for each detector element = /NXentry/NXdata/data
```

The full mapping of WONI data into NeXus is documented in the next table:

WONI data	NeXus path
title of measurement	`/NXentry/title`
sample *name*	`/NXentry/NXsample/name`
sample *temperature*	`/NXentry/NXsample/temperature`
monitor	`/NXentry/NXmonitor/data`
type of radiation *probe*	`/NXentry/MXinstrument/NXsource/probe`
wavelength of radiation incident on sample	`/NXentry/MXinstrument/NXcrystal/wavelength`
two theta of detector elements	`/NXentry/NXinstrument/NXdetector/polar_angle`
counts for each detector element	`/NXentry/NXinstrument/NXdetector/data`
two theta of detector elements	`/NXentry/NXdata/polar_angle`
counts for each detector element	`/NXentry/NXdata/data`

Looking at this table, one might get concerned that the two theta and counts data is stored in two places and thus duplicated. Stop worrying, this problem is solved at the NeXus API level. Typically `NXdata` will only hold links to the corresponding data items in `/NXentry/NXinstrument/NXdetector`.

In this step problems might occur. The first is that the base class definitions contain a bewildering number of parameters. This is on purpose: the base classes serve as dictionaries which define names for everything which possibly can occur. You do not have to give all that information. The key question is, as already said, *What is required for typical*

data analysis for this type of application? You might also be unsure how to correctly store a particular data item. In such a case, contact the NIAC for help. Another problem which can occur is that you require to store information for which there is no name in one of the existing base classes or you have a new instrument component for which there is no base class alltogether. In such a case, please feel free to contact the NIAC with a suggestion for an extension of the base classes in question.

Step 3: *Describe* this map in a NXDL file

This is even easier. Some XML editing is necessary. Fire up your XML editor of choice and open a file. If your XML editor supports XML schema while editing XML, it is worth to load `nxdl.xsd`. Now your XML editor can help you to create a proper NXDL file. As always, the start is an empty template file. This looks like the XML code below.

Note: This is just the basic XML for a NXDL definition. It is advisable to change some of the documentation strings.

NXDL template file

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
# NeXus - Neutron and X-ray Common Data Format
#
# Copyright (C) 2008-2012 NeXus International Advisory Committee (NIAC)
#
# This library is free software; you can redistribute it and/or
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation; either
# version 3 of the License, or (at your option) any later version.
#
# This library is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public
# License along with this library; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# For further information, see http://www.nexusformat.org

############ SVN repository information ###################
# $Date: 2012-05-28 22:10:09 +0100 (Mon, 28 May 2012) $
# $Author: Pete Jemian $
# $Revision: 1091 $
# $HeadURL: file:///isis/svn/nexus/definitions/trunk/manual/source/examples/NX__template__.nxdl.xml $
# $Id: NX__template__.nxdl.xml 1091 2012-05-28 21:10:09Z Pete Jemian $
############ SVN repository information ###################
-->
<definition name="NX__template__" extends="NXobject" type="group"
    category="application"
    xmlns="http://definition.nexusformat.org/nxdl/3.1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://definition.nexusformat.org/nxdl/3.1 ../nxdl.xsd"
    version="1.0b"
    >
```

```
    <doc>template for a NXDL application definition</doc>
</definition>
```

For example, copy and rename the file to `NXwoni.nxdl.xml`. Then, locate the XML root element `definition` and change the `name attribute` (the XML shorthand for this attribute is `/definition/@name`) to `NXwoni`. Change the `doc` as well. Also consider keeping track of `/definition/@version` as suits your development of this NXDL file.

The next thing which needs to be done is adding groups into the definition. A group is defined by some XML, as in this example:

```
<group type="NXdata">

</group>
```

The type is the actual NeXus base class this group belongs to. Optionally a `name` attribute may be given (default is `data`).

Next, one needs to include data items too. The XML for such a data item looks similar to this:

```
<field name="polar_angle" type="NX_FLOAT units="NX_ANGLE">
  <doc>Link to polar angle in /NXentry/NXinstrument/NXdetector</doc>
  <dimensions rank="1">
    <dim index="1" value="ndet"/>
  </dimensions>
</field>
```

The meaning of the `name` attribute is intuitive, the `type` can be looked up in the relevant base class definition. A `field` definition can optionally contain a `doc` element which contains a description of the data item. The `dimensions` entry specifies the dimensions of the data set. The `size` attribute in the dimensions tag sets the rank of the data, in this example: `rank="1"`. In the `dimensions` group there must be *rank* `dim` fields. Each `dim` tag holds two attributes: `index` determines to which dimension this tag belongs, the `1` means the first dimension. The `value` attribute then describes the size of the dimension. These can be plain integers, variables, such as in the example `ndet` or even expressions like `tof+1`.

Thus a NXDL file can be constructed. The full NXDL file for the WONI example is given in *Full listing of the WONI Application Definition*. Clever readers may have noticed the strong similarity between our working example `NXwoni` and `NXmonopd` since they are essentially identical. Give yourselves a cookie if you spotted this.

Step 4: *Standardize* with the NIAC

Basically you are done. Your first application definition for NeXus is constructed. In order to make your work a standard for that particular application type, some more steps are required:

- Send your application definition to the NIAC for review
- Correct your definition per the comments of the NIAC
- Cure and use the definition for a year
- After a final review, it becomes the standard

The NIAC must review an application definition before it is accepted as a standard. The one year curation period is in place in order to gain practical experience with the definition and to sort out bugs from Step 1. In this period, data shall be written and analyzed using the new application definition.

Full listing of the WONI Application Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="nxdlformat.xsl" ?>
<!--
# NeXus - Neutron and X-ray Common Data Format
#
# Copyright (C) 2008-2012 NeXus International Advisory Committee (NIAC)
#
# This library is free software; you can redistribute it and/or
# modify it under the terms of the GNU Lesser General Public
# License as published by the Free Software Foundation; either
# version 3 of the License, or (at your option) any later version.
#
# This library is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
# Lesser General Public License for more details.
#
# You should have received a copy of the GNU Lesser General Public
# License along with this library; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
#
# For further information, see http://www.nexusformat.org

############ SVN repository information ###################
# $Date: 2012-09-28 17:47:09 +0100 (Fri, 28 Sep 2012) $
# $Author: Pete Jemian $
# $Revision: 1172 $
# $HeadURL: file:///isis/svn/nexus/definitions/trunk/applications/NXmonopd.nxdl.xml $
# $Id: NXmonopd.nxdl.xml 1172 2012-09-28 16:47:09Z Pete Jemian $
############ SVN repository information ###################
-->
<definition name="NXmonopd" extends="NXobject" type="group"
    category="application"
    xmlns="http://definition.nexusformat.org/nxdl/@NXDL_RELEASE@"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://definition.nexusformat.org/nxdl/@NXDL_RELEASE@ ../nxdl.xsd"
    version="1.0b"
    svnid="$Id: NXmonopd.nxdl.xml 1172 2012-09-28 16:47:09Z Pete Jemian $">
    <doc>
        Monochromatic Neutron and X-Ray Powder Diffraction. Instrument
        definition for a powder diffractometer at a monochromatic neutron
        or X-ray beam. This is both suited for a powder diffractometer
        with a single detector or a powder diffractometer with a position
        sensitive detector.
    </doc>
    <group type="NXentry" name="entry">
        <field name="title"/>
        <field name="start_time" type="NX_DATE_TIME"/>
        <field name="definition">
            <doc> Official NeXus NXDL schema to which this file conforms </doc>
            <enumeration>
                <item value="NXmonopd"/>
            </enumeration>
        </field>
        <group type="NXinstrument">
            <group type="NXsource">
```

```
            <field name="type"/>
            <field name="name"/>
            <field name="probe">
                <enumeration>
                    <item value="neutron"/>
                    <item value="x-ray"/>
                    <item value="electron"/>
                </enumeration>
            </field>
        </group>
        <group type="NXcrystal">
            <field name="wavelength" type="NX_FLOAT" units="NX_WAVELENGTH">
                <doc>Optimum diffracted wavelength</doc>
                <dimensions rank="1">
                    <dim index="1" value="i"/>
                </dimensions>
            </field>
        </group>
        <group type="NXdetector">
            <field name="polar_angle" type="NX_FLOAT" axis="1">
              <doc>where ndet = number of detectors</doc>
              <dimensions rank="1">
                <dim index="1" value="ndet" />
              </dimensions>
            </field>
            <field name="data" type="NX_INT" signal="1">
              <doc>
                detector signal (usually counts) are already
                corrected for detector efficiency
              </doc>
              <dimensions rank="1">
                <dim index="1" value="ndet" />
              </dimensions>
            </field>
        </group>
    </group>
    <group type="NXsample">
        <field name="name">
            <doc>Descriptive name of sample</doc>
        </field>
        <field name="rotation_angle" type="NX_FLOAT" units="NX_ANGLE">
            <doc>
                Optional rotation angle for the case when the powder diagram
                has been obtained through an omega-2theta scan like from a
                traditional single detector powder diffractometer
            </doc>
        </field>
    </group>
    <group type="NXmonitor">
        <field name="mode">
            <doc>
                Count to a preset value based on either clock time (timer)
                or received monitor counts (monitor).
            </doc>
            <enumeration>
                <item value="monitor"/>
                <item value="timer"/>
            </enumeration>
```

```
            </field>
            <field name="preset" type="NX_FLOAT">
                <doc>preset value for time or monitor</doc>
            </field>
            <field name="integral" type="NX_FLOAT" units="NX_ANY">
                <doc>Total integral monitor counts</doc>
            </field>
        </group>
        <group type="NXdata">
            <link name="polar_angle" target="/NXentry/NXinstrument/NXdetector/polar_angle">
                <doc>Link to polar angle in /NXentry/NXinstrument/NXdetector</doc>
            </link>
            <link name="data" target="/NXentry/NXinstrument/NXdetector/data">
                <doc>Link to data in /NXentry/NXinstrument/NXdetector</doc>
            </link>
        </group>
    </group>
</definition>
```

Using an Application Definition

The application definition is like an interface for your data file. In practice files will contain far more information. For this, the extendable capability of NeXus comes in handy. More data can be added, and upstream software relying on the interface defined by the application definition can still retrieve the necessary information without any changes to their code.

NeXus application definitions only standardize classes. You are free to decide upon names of groups, subject to them matching regular expression for NeXus name attributes (see the *regular expression pattern for NXDL group and field names* in the *Naming Conventions* section). Note the length limit of 63 characters imposed by HDF5. Please use sensible, descriptive names and separate multi worded names with underscores.

Something most people wish to add is more metadata, for example in order to index files into a database of some sort. Go ahead, do so, if applicable, scan the NeXus base classes for standardized names. For metadata, consider to use the `NXarchive` definition. In this context, it is worth to mention that a practical NeXus file might adhere to more then one application definition. For example, WONI data files may adhere to both the `NXmonopd` and `NXarchive` definitions. The first for data analysis, the second for indexing into the database.

Often, instrument scientists want to store the complete state of their instrument in data files in order to be able to find out what went wrong if the data is unsatisfactory. Go ahead, do so, please use names from the NeXus base classes.

Site policy might require you to store the names of all your bosses up to the current head of state in data files. Go ahead, add as many `NXuser` classes as required to store that information. Knock yourselves silly over this.

Your Scientific Accounting Department (SAD) may ask of you the preposterous; to store billing information into data files. Go ahead, do so if your judgment allows. Just do not expect the NIAC to provide base classes for this and do not use the prefix NX for your classes.

In most cases, NeXus files will just have one `NXentry` class group. But it may be required to store multiple related data sets of the results of data analysis into the same data file. In this case create more entries. Each entry should be interpretable standalone, i.e. contain all the information of a complete `NXentry` class. Please keep in mind that groups or data items which stay constant across entries can always be linked in.

1.4.4 Processed Data

Data reduction and analysis programs are encouraged to store their results in NeXus data files. As far as the necessary, the normal NeXus hierarchy is to be implemented. In addition, processed data files must contain a *NXprocess* group.

This group, that documents and preserves data provenance, contains the name of the data processing program and the parameters used to run this program in order to achieve the results stored in this entry. Multiple processing steps must have a separate entry each.

1.5 Strategies for storing information in NeXus data files

NeXus may appear daunting, at first, to use. The number of base classes is quite large as well as is the number of application definitions. This chapter describes some of the strategies that have been recommended for how to store information in NeXus data files.

When we use the term *storing*, some might be helped if they consider this as descriptions for how to *classify* their data.

It is intended for this chapter to grow, with the addition of different use cases as they are presented for suggestions.

1.5.1 Strategies: The simplest case(s)

Perhaps the simplest case might be either a step scan with two or more columns of data. Another simple case might be a single image acquired by an area detector. In either of these hypothetical cases, the situation is so simple that there is little addition information available to be described (for whatever reason).

Step scan with two or more data columns

Consider the case where we wish to store the data from a step scan. This case may involve two or more *related* 1-D arrays of data to be saved, each having the same length. For our hypothetical case, we'lll have these positioners as arrays:

positioner arrays	detector arrays
`ar`, `ay`, `dy`	`I0`, `I00`, `time`, `Epoch`, `photodiode`

1.5.2 Strategies: The wavelength

Where should the wavelength of my experiment be written? This is one of the *Frequently Asked Questions*. The canonical location to store wavelength has been:

```
/NXentry/NXinstrument/NXcrystal/wavelength
```

More recently, this location makes more sense to many:

```
/NXentry/NXinstrument/NXmonochromator/wavelength
```

NXcrystal describes a crystal monochromator or analyzer. Recently, scientists with monochromatic radiation not defined by a crystal, such as from an electron-beam undulator or a neutron helical velocity selector, were not satisfied with creating a fictitious instance of a crystal just to preserve the wavelength from their instrument. Thus, the addition of the *NXmonochromator* base class to NeXus, which also allows "energy" to be specified if one is so inclined.

Note: See the *Class path specification* section for a short discussion of the difference between the HDF5 path and the NeXus symbolic class path.

1.5.3 Strategies: The next case

The *NIAC: The NeXus International Advisory Committee* welcomes suggestions for additional sections in this chapter.

1.6 Verification and validation of files

The intent of verification and validation of files is to ensure, in an unbiased way, that a given file conforms to the relevant specifications. NeXus uses various automated tools to validate files. These tools include conversion of content from HDF to XML and transformation (via XSLT) from XML format to another such as NXDL, XSD, and Schematron. This chapter will first provide an overview of the process, then define the terms used in validation, then describe how multiple base classes or application definitions might apply to a given NeXus data file, and then describe the various validation techniques in more detail. Validation does not check that the data content of the file is sensible; this requires scientific interpretation based on the technique.

Validation is useful to anyone who manipulates or modifies the contents of NeXus files. This includes scientists/users, instrument staff, software developers, and those who might mine the files for metadata. First, the scientist or user of the data must be certain that the information in a file can be located reliably. The instrument staff or software developer must be confident the information they have written to the file has been located and formatted properly. At some time, the content of the NeXus file may contribute to a larger body of work such as a metadata catalog for a scientific instrument, a laboratory, or even an entire user facility.

1.6.1 Overview

NeXus files adhere to a set of rules and can be tested against these rules for compliance. The rules are implemented using standard tools and can themselves be tested to verify compliance with the standards for such definitions. Validation includes the testing of both NeXus data files and the NXDL specifications that describe the rules.

The rules for writing NeXus data files are different than the rules for writing NeXus class definitions. To validate a NeXus data file, these two rule sets must eventually merge, as shown in the next figure. The data file (either HDF4, HDF5, or XML) is first converted into an internal format to facilitate validation, including data types, array dimensions, naming, and other items. Most of the data is not converted since data validation is non-trivial. Also note that the units are not validated. All the NXDL files are converted into a single Schematron file (again, internal use for validation) only when NXDL revisions are checked into the NeXus definitions repository as NXDL changes are not so frequent.

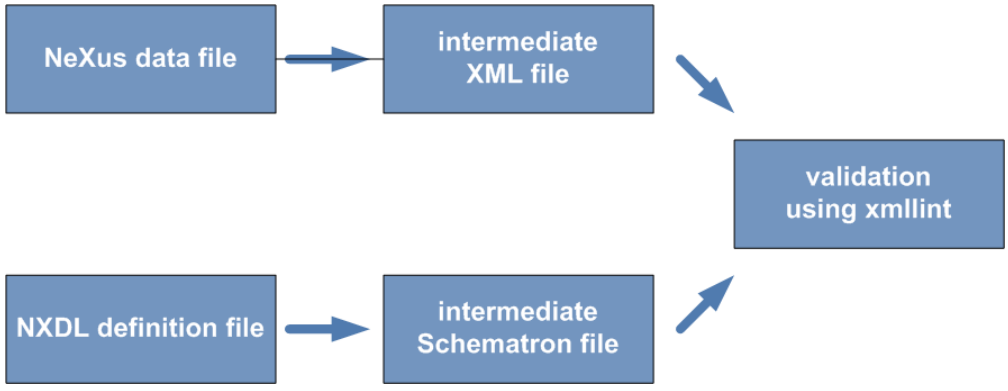


Figure 1.9: Flowchart of the NeXus validation process.

NeXus data files NeXus data files (also known as NeXus data file instances) are validated to ensure the various parts of the data file are arranged according to the governing NXDL specifications used in that file instance.

Note: Since NeXus has several rules that are quite difficult to apply in either XSD or Schematron, direct validation of data files using standard tools is not possible. To validate NeXus data files, it is necessary to use `nxvalidate`.

NeXus Definition Language (NXDL) specification files NXDL files are validated to ensure they adhere to the rules for writing NeXus base classes and application definitions.

1.6.2 Definitions of these terms

Let's be clear about some terms used in this section.

HDF Hierarchical Data Format from The HDF Group. NeXus data files using HDF may be stored in either version 4 (HDF4) or version 5 (HDF5). New NeXus HDF files should only use HDF5. The preferred file extensions (but not required) include `.hdf`, `.h5`, `.nxs`, and `.nx5`.

NXDL NeXus Definition Language files define the specifications for NeXus base classes, application definitions, and contributed classes and definitions. It is fully described in the *NXDL* chapter.

Schematron Schematron [13] is an alternative to XSD and is used to validate the content and structure of an XML file. NeXus uses Schematron internally to validate data files.

Validation File validation is the comparison of file contents, in an unbiased way, with the set of rules that define the structure of such files.

XML The eXtensible Markup Language (XML) [14] is a standard business tool for the exchange of information. It is broadly supported by a large software library in many languages. NeXus uses XML for several purposes: data files, NXDL definitions, rules, and XSLT transformations.

XSD XML files are often defined by a set of rules (or *schema*). A common language used to implement these rules is XML Schema (XSD) [15] Fundamentally, all XML, XSD, XSLT, and Schematron files are XML.

XSLT XML files can be flexible enough to convert from one set of rules to another. An example is when one company wishes to exchange catalog or production information with another. The XML StyLsheet Transformation (XSLT) [16] language is often used to describe each direction of the conversion of the XML files between the two rule sets.

1.6.3 NeXus data files may use multiple base classes or application definitions

NeXus data files may have more than one data set or may have multiple instances of just about any base class or even application definitions. The NeXus data file validation is prepared to handle this without any special effort by the provider of the data file.

1.6.4 Validation techniques

File validation is the process to determine if a given file is prepared consistent with a set of guidelines or rules. In NeXus, there are several different types of files. First, of course, is the data file yet it can be provided in one of several forms: HDF4, HDF5, or XML. Specifications for data files are provided by one or (usually) more NeXus definition files (NXDL, for short). These NXDL files are written in XML and validated by the NXDL specification which is

[13] http://www.schematron.com
[14] http://www.w3schools.com/xml
[15] http://www.w3schools.com/schema
[16] http://www.w3schools.com/xsl/

written in the XML Schema (XSD) language. Thus, automated file verification is available for data files, definition files, and the rules for definition files.

Validation of NeXus data files

Each NeXus data file can be validated against the NXDL rules. (The full suite of NXDL specifications is converted into Schematron rules by an XSLT transformation and then combined into a single file. It is not allowed to have a NeXus base class and also an application definition with the same name since one will override the other in the master Schematron file) The validation is done using Schematron and the `NXvalidate` program. Schematron was selected, rather than XML Schema (XSD), to permit established rules for NeXus files, especially the rule allowing the nodes within `NXentry` to appear in any order.

The validation process is mainly checking file structure (presence or absence of groups/fields) - it is usually impossible to check the actual data itself, other than confirm that it is of the correct data type (string, float etc.). The only exception is when the NXDL specification is either a fixed value or an enumeration - in which case the data is checked.

During validation, the NeXus data file instance (either HDF or XML) is first converted into an XML file in a form that facilitates validation (e.g with large numeric data removed). Then the XML file is validated by Schematron against the `schema/all.sch` file.

Validation of NeXus Definition Language (NXDL) specification files

Each NXDL file must be validated against the rules that define how NXDL files are to be arranged. The NXDL rules are specified in the form of XML Schema (XSD).

Standard tools (validating editor or command line or support library) can be used to validate any NXDL file. Here's an example using `xmllint` from a directory that contains `nxdl.xsd`, `nxdlTypes.xsd`, and `applications/NXsas.nxdl.xml`:

Use of `xmllint` to validate a NXDL specification.

```
xmllint --noout --schema nxdl.xsd applications/NXsas.nxdl.xml
```

Validation of the NXDL rules

NXDL rules are specified using the rules of XML Schema (XSD). The XSD syntax of the rules is validated using standard XML file validation tools: either a validating editor (such as *oXygen*, *xmlSpy*, or *eclipse*) or common UNIX/Linux command line tools

Use of `xmllint` to validate the NXDL rules.

```
xmllint --valid  nxdl.xsd
```

The validating editor method is used by the developers while the `xmllint` command line tool is the automated method used by the NeXus definitions subversion repository.

Validation of XSLT files

XSLT transformations are validated using standard tools such as a validating editor or xmllint.

Transformation of NXDL files to Schematron

Schematron [1] is a rule-based language that allows very specific validation of an XML document. Its advantages over using XSD schema are that:

- more specific pattern-based rules based on data content can be written
- full XSLT/XPath expression syntax available for writing validation tests
- error messages can be customised and thus more meaningful
- It is easier to validate documents when entities can occur in any order.

XSD does provide a mechanism for defining a class structure and inheritance, so its usage within NeXus in addition to schematron has not been ruled out. But for a basic validation of file content, schematron looks best.

The NXDL definition files are converted into a set of Schematron rules using the `xslt/nxdl2sch.xsl` XSLT stylesheet. The NeXus instance file (either in XML, HDF4, or HDF5) is turned into a reduced XML validation file. This file is very similar to a pure NeXus XML file, but with additional metadata for dimensions and also with most of the actual numeric data removed.

The validation process then compares the set of Schematron rules against the *reduced XML* validation file. Schematron itself is implemented as a set of XSLT transforms. NeXus includes the Schematron files, as well as the Java based XSLT engine `saxon`.

The java based `nxvalidate` GUI can be run to validate files.

Currently, the structure of the file is validated (i.e. valid names are used at the correct points), but this will be extended to array dimensions and link targets. Error messages are printed about missing mandatory fields, and informational messages are printed about fields that are neither optional or mandatory (in case they are a typing error). Even non-standard names must comply with a set of rules (e.g. no spaces are allowed in names). Enumerations are checked that they conform to an allowed value. The data type is checked and the units will also be checked.

CHAPTER

TWO

NEXUS: REFERENCE DOCUMENTATION

2.1 NAPI: NeXus Application Programmer Interface

The Application Program Interface (API) has been developed to facilitate the reading and writing of NeXus files. Those writing applications to produce NeXus files are encouraged to use the API in order to ensure compliance with the NeXus standard. The API supports the reading and writing of HDF4, HDF5, and XML files. However, it is possible to create programs that write and/or read NeXus files without using the NAPI. Two examples are provided in *Example NeXus C programs using native HDF5 commands*.

The core routines have been written in C but wrappers are available for a number of other languages including C++, Fortran 77, Fortran 90, Java, Python and IDL. The API makes the reading and writing of NeXus files transparent; the user doesn't even need to know the underlying format when reading a file since the API calls are the same.

More in-depth and up-to-date information about the NeXus Application Programming Interface for the various language backends is available on-line from http://download.nexusformat.org.

The NeXusIntern.pdf document (http://svn.nexusformat.org/code/trunk/doc/api/NeXusIntern.pdf) describes the internal workings of the NeXus-API. You are very welcome to read it, but it will not be of much use if all you want is to read and write files using the NAPI.

2.1.1 Purpose of API

The NeXus Application Program Interface call routines in the appropriate backend (HDF4, HDF5 or XML) to read and write files with the correct structure. The API serves a number of purposes:

1. It simplifies the reading and writing of NeXus files.
2. It ensures a certain degree of compliance with the NeXus standard.
3. It hides the implementation details of the format. In particular, the API can read and write HDF4, HDF5, and XML files using the same routines.

Note: While it is possible to write NeXus compilant data files using a supported on-disk backend directly, we strongly encourage using the supported NeXus-API. This ensures the maximum compatibily.

2.1.2 Core API

The core API provides the basic routines for reading, writing and navigating NeXus files. Operations are performed using a handle that keeps a record of its current position in the file hierarchy. All are read or write requests are then implicitly performed on the currently *open* entity. This limits number of parameters that need to be passed to API calls, at the cost of forcing a certain mode of operation. It is very similar to navigating a directory hierarchy; NeXus groups are the directories, which can contain data sets and/or other directories.

The core API comprises the following functional groups:

- General initialization and shutdown: opening and closing the file, creating or opening an existing group or dataset, and closing them.
- Reading and writing data and attributes to previously opened datasets.
- Routines to obtain meta-data and to iterate over component datasets and attributes.
- Handling of linking and group hierarchy.
- Routines to handle memory allocation. (Not required in all language bindings.)

NAPI C and C++ Interface

Doxygen documentation is provided online:

C http://download.nexusformat.org/doxygen/html-c/

C++ http://download.nexusformat.org/doxygen/html-cpp/

NAPI Fortran 77 Interface

Doxygen documentation is provided for the f77 NAPI. (http://download.nexusformat.org/doxygen/html-f77/)

NAPI Fortran 90 Interface

The Fortran 90 interface is a wrapper to the C interface with nearly identical routine definitions. As with the Fortran 77 interface, it is necessary to reverse the order of indices in multidimensional arrays, compared to an equivalent C program, so that data are stored in the same order in the NeXus file.

Any program using the F90 API needs to put the following line at the top (after the `PROGRAM` statement):

```
use NXmodule
```

Use the following table to convert from the C data types listed with each routine to the Fortran 90 data types.

C data type	F90 data type
`int, int`	`integer`
`char*`	`character(len=*)`
`NXhandle, NXhandle*`	`type(NXhandle)`
`NXstatus`	`integer`
`int[]`	`integer(:)`
`void*`	`real(:)` or `integer(:)` or `character(len=*)`
`NXlink a, NXlink* a`	`type(NXlink)`

The parameters in the next table, defined in `NXmodule`, may be used in defining variables.

Name	Description	Value
`NX_MAXRANK`	Maximum number of dimensions	`32`
`NX_MAXNAMELEN`	Maximum length of NeXus name	`64`
`NXi1`	Kind parameter for a 1-byte integer	`selected_int_kind(2)`
`NXi2`	Kind parameter for a 2-byte integer	`selected_int_kind(4)`
`NXi4`	Kind parameter for a 4-byte integer	`selected_int_kind(8)`
`NXr4`	Kind parameter for a 4-byte real	`kind(1.0)`
`NXr8`	Kind parameter for an 8-byte real	`kind(1.0D0)`

Also see the doxygen documentation. (http://download.nexusformat.org/doxygen/html-f90/)

NAPI Java Interface

This section includes installation notes, instructions for running NeXus for Java programs and a brief introduction to the API.

The Java API for NeXus (`jnexus`) was implemented through the Java Native Interface (JNI) to call on to the native C library. This has a number of disadvantages over using pure Java, however the most popular file backend HDF5 is only available using a JNI wrapper anyway.

Acknowledgement

This implementation uses classes and native methods from NCSA's Java HDF Interface project. Basically all conversions from native types to Java types is done through code from the NCSA HDF group. Without this code the implementation of this API would have taken much longer. See NCSA's copyright for more information.

Installation

Requirements

> **Caution:** Documentation is old and may need revision.

For running an application with `jnexus` an recent Java runtime environment (JRE) will do.

In order to compile the Java API for NeXus a Java Development Kit is required on top of the build requirements for the C API.

Installation under Windows

1. Copy the HDF DLL's and the file `jnexus.dll` to a directory in your path. For instance `C:\\Windows\\system32`.
2. Copy the `jnexus.jar` to the place where you usually keep library jar files.

Note that the location or the naming of these files in the binary Nexus distributions have changed over the years. In the Nexus 4.3.0 Windows 64-bit distribution (http://download.nexusformat.org/kits/4.3.0/win64/), By default, the DLL is at: `C:\Program Files\NeXus Data Format\bin\libjnexus-0.dll`. Please rename this file to `jnexus.dll` before making it available in your path. This is important, otherwise, JVM runtime will not be able to locate this file.

For the same distribution, the location of `jnexus.jar` is at: `C:\Program Files\NeXus Data Format\share\java`.

Installation under Unix The `jnexus.so` shared library as well as all required file backend `.so` libraries are required as well as the `jnexus.jar` file holding the required Java classes. Copy them wherever you like and see below for instructions how to run programs using `jnexus`.

Running Programs with the NeXus API for Java

In order to successfully run a program with `jnexus`, the Java runtime systems needs to locate two items:

1. The shared library implementing the native methods.
2. The `nexus.jar` file in order to find the Java classes.

Locating the shared libraries The methods for locating a shared library differ between systems. Under Windows32 systems the best method is to copy the `jnexus.dll` and the HDF4, HDF5 and/or XML-library DLL files into a directory in your path.

On a UNIX system, the problem can be solved in three different ways:

1. Make your system administrator copy the `jnexus.so` file into the systems default shared library directory (usually `/usr/lib` or `/usr/local/lib`).
2. Put the `jnexus.so` file wherever you see fit and set the `LD_LIBRARY_PATH` environment variable to point to the directory of your choice.
3. Specify the full pathname of the jnexus shared library on the java command line with the `-Dorg.nexusformat.JNEXUSLIB=full-path-2-shared-library` option.

Locating `jnexus.jar` This is easier, just add the the full pathname to `jnexus.jar` to the classpath when starting java. Here are examples for a UNIX shell and the Windows shell.

UNIX example shell script to start `jnexus.jar`

```
#!/sbin/sh
java -classpath /usr/lib/classes.zip:../jnexus.jar:. \
    -Dorg.nexusformat.JNEXUSLIB=../libjnexus.so TestJapi
```

Windows 32 example batch file to start `jnexus.jar`

```
set JL=-Dorg.nexusformat.JNEXUSLIB=..\jnexus\bin\win32\jnexus.dll
java -classpath C:\jdk1.5\lib\classes.zip;..\jnexus.jar;. %JL% TestJapi
```

Programming with the NeXus API for Java

The NeXus C-API is good enough but for Java a few adaptions of the API have been made in order to match the API better to the idioms used by Java programmers. In order to understand the Java-API, it is useful to study the NeXus C-API because many methods work in the same way as their C equivalents. A full API documentation is available in Java documentation format. For full reference look especially at:

- The interface `NeXusFileInterface` first. It gives an uncluttered view of the API.
- The implementation `NexusFile` which gives more details about constructors and constants. However this documentation is interspersed with information about native methods which should not be called by an application programmer as they are not part of the standard and might change in future.

See the following code example for opening a file, opening a vGroup and closing the file again in order to get a feeling for the API:

fragment for opening and closing

```
1      // $Id: napi-java-prog1.java 1091 2012-05-28 21:10:09Z Pete Jemian $
2      try{
3          NexusFile nf = new NexusFile(filename, NexusFile.NXACC_READ);
4          nf.opengroup("entry1","NXentry");
5          nf.finalize();
6      }catch(NexusException ne) {
7          // Something was wrong!
8      }
```

Some notes on this little example:

- Each NeXus file is represented by a `NexusFile` object which is created through the constructor.
- The `NexusFile` object takes care of all file handles for you. So there is no need to pass in a handle anymore to each method as in the C language API.
- All error handling is done through the Java exception handling mechanism. This saves all the code checking return values in the C language API. Most API functions return void.
- Closing files is tricky. The Java garbage collector is supposed to call the finalize method for each object it decides to delete. In order to enable this mechanism, the `NXclose()` function was replaced by the `finalize()` method. In practice it seems not to be guaranteed that the garbage collector calls the `finalize()` method. It is safer to call `finalize()` yourself in order to properly close a file. Multiple calls to the `finalize()` method for the same object are safe and do no harm.

Data Writing and Reading

Again a code sample which shows how this looks like:

fragment for writing and reading

```
1      // $Id: napi-java-datarw1.java 1091 2012-05-28 21:10:09Z Pete Jemian $
2      int idata[][] = new idata[10][20];
3      int iDim[] = new int[2];
4
5      // put some data into idata.......
6
```

```
    // write idata
    iDim[0] = 10;
    iDim[1] = 20;
    nf.makedata("idata",NexusFile.NX_INT32,2,iDim);
    nf.opendata("idata");
    nf.putdata(idata);

    // read idata
    nf.getdata(idata);
```

The dataset is created as usual with `makedata()` and opened with `putdata()`. The trick is in `putdata()`. Java is meant to be type safe. One would think then that a `putdata()` method would be required for each Java data type. In order to avoid this, the data to `write()` is passed into `putdata()` as type `Object`. Then the API proceeds to analyze this object through the Java introspection API and convert the data to a byte stream for writing through the native method call. This is an elegant solution with one drawback: An array is needed at all times. Even if only a single data value is written (or read) an array of length one and an appropriate type is the required argument.

Another issue are strings. Strings are first class objects in Java. HDF (and NeXus) sees them as dumb arrays of bytes. Thus strings have to be converted to and from bytes when reading string data. See a writing example:

String writing

```
    // $Id: napi-java-datarw2.java 1091 2012-05-28 21:10:09Z Pete Jemian $
    String ame = "Alle meine Entchen";
    nf.makedata("string_data",NexusFile.NX_CHAR,
            1,ame.length()+2);
    nf.opendata("string_data");
    nf.putdata(ame.getBytes());
```

And reading:

String reading

```
    // $Id: napi-java-datarw2.java 1091 2012-05-28 21:10:09Z Pete Jemian $
    String ame = "Alle meine Entchen";
    nf.makedata("string_data",NexusFile.NX_CHAR,
            1,ame.length()+2);
    nf.opendata("string_data");
    nf.putdata(ame.getBytes());
```

The aforementioned holds for all strings written as SDS content or as an attribute. SDS or vGroup names do not need this treatment.

Inquiry Routines

Let us compare the C-API and Java-API signatures of the `getinfo()` routine (C) or method (Java):

C API signature of `getinfo()`

```
    /* $Id: frag-c-api-sig-getinfo.c 1091 2012-05-28 21:10:09Z Pete Jemian $ */
    /* C -API */
```

```
    NXstatus NXgetinfo(NXhandle handle, int *rank, int iDim[],
                       int *datatype);
```

Java API signature of `getinfo()`

```
    // $Id: frag-c-api-sig-getinfo.java 1091 2012-05-28 21:10:09Z Pete Jemian $
    // Java
    void getinfo(int iDim[], int args[]);
```

The problem is that Java passes arguments only by value, which means they cannot be modified by the method. Only array arguments can be modified. Thus `args` in the `getinfo()` method holds the rank and datatype information passed in separate items in the C-API version. For resolving which one is which, consult a debugger or the API-reference.

The attribute and vGroup search routines have been simplified using Hashtables. The `Hashtable` returned by `groupdir()` holds the name of the item as a key and the classname or the string SDS as the stored object for the key. Thus the code for a vGroup search looks like this:

vGroup search

```
    // $Id: napi-java-inquiry1.java 1091 2012-05-28 21:10:09Z Pete Jemian $
    nf.opengroup(group,nxclass);
    h = nf.groupdir();
    e = h.keys();
    System.out.println("Found in vGroup entry:");
    while(e.hasMoreElements())
    {
        vname = (String)e.nextElement();
        vclass = (String)h.get(vname);
        System.out.println("     Item: " + vname + " class: " + vclass);
    }
```

For an attribute search both at global or SDS level the returned Hashtable will hold the name as the key and a little class holding the type and size information as value. Thus an attribute search looks like this in the Java-API:

attribute search

```
    // $Id: napi-java-inquiry2.java 1091 2012-05-28 21:10:09Z Pete Jemian $
    Hashtable h = nf.attrdir();
    Enumeration e = h.keys();
    while(e.hasMoreElements())
    {
        attname = (String)e.nextElement();
        atten = (AttributeEntry)h.get(attname);
        System.out.println("Found global attribute: " + attname +
            " type: "+ atten.type + " ,length: " + atten.length);
    }
```

For more information about the usage of the API routines see the reference or the NeXus C-API reference pages. Another good source of information is the source code of the test program which exercises each API routine.

Known Problems

These are a couple of known problems which you might run into:

Memory As the Java API for NeXus has to convert between native and Java number types a copy of the data must be made in the process. This means that if you want to read or write 200MB of data your memory requirement will be 400MB! This can be reduced by using multiple `getslab()`/`putslab()` to perform data transfers in smaller chunks.

`Java.lang.OutOfMemoryException` By default the Java runtime has a low default value for the maximum amount of memory it will use. This ceiling can be increased through the `-mxXXm` option to the Java runtime. An example: `java -mx512m ...` starts the Java runtime with a memory ceiling of 512MB.

Maximum 8192 files open The NeXus API for Java has a fixed buffer for file handles which allows only 8192 NeXus files to be open at the same time. If you ever hit this limit, increase the `MAXHANDLE` define in `native/handle.h` and recompile everything.

On-line Documentation

The following documentation is browsable online:

1. The Doxygen API documentation [1]
2. A verbose tutorial for the NeXus for Java API.
3. The API Reference.
4. Finally, the source code for the test driver for the API which also serves as a documented usage example.

NAPI Python Interface

Documentation available in pydoc and doxygen. (http://download.nexusformat.org/doxygen/html-python)

NAPI IDL Interface

IDL is an interactive data evaluation environment developed by Research Systems - it is an interpreted language for data manipulation and visualization. The NeXus IDL bindings allow access to the NeXus API from within IDL - they are installed when NeXus is compiled from source after being configured with the following options:

```
configure \
        --with-idlroot=/path/to/idl/installation \
        --with-idldlm=/path/to/install/dlm/files/to
```

For further details see the README (http://svn.nexusformat.org/code/trunk/bindings/idl/README.html) for the NeXus IDL binding.

2.1.3 Utility API

The NeXus F90 Utility API provides a number of routines that combine the operations of various core API routines in order to simplify the reading and writing of NeXus files. At present, they are only available as a Fortran 90 module but a C version is in preparation.

The utility API comprises the following functional groups:

[1] http://download.nexusformat.org/doxygen/html-java/

- Routines to read or write data.
- Routines to find whether or not groups, data, or attributes exist, and to find data with specific signal or axis attributes, i.e. to identify valid data or axes.
- Routines to open other groups to which `NXdata` items are linked, and to return again.

line required for use with F90 API

Any program using the F90 Utility API needs to put the following line near the top of the program:

```
use NXUmodule
```

Note: Do not put `USE` statements for both `NXmodule` and `NXUmodule`. The former is included in the latter

---

List of F90 Utility Routines

name	description
Reading and Writing	
`NXUwriteglobals`	Writes all the valid global attributes of a file.
`NXUwritegroup`	Opens a group (creating it if necessary).
`NXUwritedata`	Opens a data item (creating it if necessary) and writes data and its units.
`NXUreaddata`	Opens and reads a data item and its units.
`NXUwritehistogram`	Opens one dimensional data item (creating it if necessary) and writes histogram centers and their units.
`NXUreadhistogram`	Opens and reads a one dimensional data item and converts it to histogram bin boundaries.
`NXUsetcompress`	Defines the compression algorithm and minimum dataset size for subsequent write operations.
Finding Groups, Data, and Attributes	
`NXUfindclass`	Returns the name of a group of the specified class if it is contained within the currently open group.
`NXUfinddata`	Checks whether a data item of the specified name is contained within the currently open group.
`NXUfindattr`	Checks whether the currently open data item has the specified attribute.
`NXUfindsignal`	Searches the currently open group for a data item with the specified `SIGNAL` attribute.
`NXUfindaxis`	Searches the currently open group for a data item with the specified `AXIS` attribute.
Finding Linked Groups	
`NXUfindlink`	Finds another link to the specified NeXus data item and opens the group it is in.
`NXUresumelink`	Reopens the original group from which `NXUfindlink` was used.

Currently, the F90 utility API will only write character strings, 4-byte integers and reals, and 8-byte reals. It can read other integer sizes into four-byte integers, but does not differentiate between signed and unsigned integers.

2.1.4 Building Programs

The install kit provides a utility call `nxbuild` that can be used to build simple programs:

```
nxbuild -o test test.c
```

This script links in the various libraries for you and reading its contents would provide the necessary information for creating a separate Makefile. You can also use `nxbuild` with the example files in the NeXus distribution kit which are installed into `/usr/local/nexus/examples`

Note that the executable name is important in this case as the test program uses it internally to determine the `NXACC_CREATE*` argument to pass to `NXopen`.

building and running a simple NeXus program

```
#  builds HDF5 specific test
nxbuild -o napi_test-hdf5 napi_test.c

# runs the test
./napi_test-hdf5
```

NeXus is also set up for pkg-config so the build can be done as:

```
gcc `pkg-config --cflags` `pkg-config --libs` -o test test.c
```

2.1.5 Reporting Bugs in the NeXus API

If you encounter any bugs in the installation or running of the NeXus API, please report them online using our Issue Reporting system. (http://www.nexusformat.org/IssueReporting)

2.2 NXDL: The NeXus Definition Language

Information in NeXus data files is arranged by a set of rules. These rules facilitate the exchange of data between scientists and software by standardizing common terms such as the way engineering units are described and the names for common things and the way that arrays are described and stored.

The set of rules for storing information in NeXus data files is declared using the NeXus Definition Language. NXDL itself is governed by a set of rules (a *schema*) that should simplify learning the few terms in NXDL. In fact, the NXDL rules, written as an XML Schema, are machine-readable using industry-standard and widely-available software tools for XML files such as `xsltproc` and `xmllint`. This chapter describes the rules and terms from which NXDL files are constructed.

2.2.1 Introduction

NeXus Definition Language (NXDL) files allow scientists to define the nomenclature and arrangement of information in NeXus data files. These NXDL files can be specific to a scientific discipline such as tomography or small-angle scattering, specific analysis or data reduction software, or even to define another component (base class) used to design and build NeXus data files.

In addition to this chapter and the *Tutorial* chapter, look at the set of NeXus NXDL files to learn how to read and write NXDL files. These files are available from the NeXus *definitions* repository and are most easily viewed through the TRAC site: http://trac.nexusformat.org/definitions/browser/trunk in the `base_classes`, `applications`, and `contributed` directories. The rules (expressed as XML Schema) for NXDL files may also be viewed from this URL. See the files `nxdl.xsd` for the main XML Schema and `nxdlTypes.xsd` for the listings of allowed data types and categories of units allowed in NXDL files.

NXDL files can be checked (validated) for syntax and content. With validation, scientists can be certain their definitions will be free of syntax errors. Since NXDL is based on the XML standard, there are many editing programs [2] available to ensure that the files are *well-formed*. [3] There are many standard tools such as `xmllint` and `xsltproc`

[2] For example *XML Copy Editor* (http://xml-copy-editor.sourceforge.net/)
[3] http://en.wikipedia.org/wiki/XML#Well-formedness_and_error-handling

that can process XML files. Further, NXDL files are backed by a set of rules (an *XML Schema*) that define the language and can be used to check that an NXDL file is both correct by syntax and valid by the NeXus rules.

NXDL files are machine-readable. This enables their automated conversion into schema files that can be used, in combination with other NXDL files, to validate NeXus data files. In fact, all of the tables in the *Class Definitions* Chapter have been generated directly from the NXDL files.

The language of NXDL files is intentionally quite small, to provide only that which is necessary to describe scientific data structures (or to establish the necessary XML structures). Rather than have scientists prepare XML Schema files directly, NXDL was designed to reduce the jargon necessary to define the structure of data files. The two principle objects in NXDL files are: `group` and `field`. Documentation (`doc`) is optional for any NXDL component. Either of these objects may have additional `attributes` that contribute simple metadata.

The *Class Definitions* Chapter lists the various classes from which a NeXus file is constructed. These classes provide the glossary of items that could, in principle, be stored in a standard-conforming NeXus file (other items may be inserted into the file if the author wishes, but they won't be part of the standard). If you are going to include a particular piece of metadata, refer to the class definitions for the standard nomenclature. However, to assist those writing data analysis software, it is useful to provide more than a glossary; it is important to define the required contents of NeXus files that contain data from particular classes of neutron, X-ray, or muon instrument.

NXDL Elements and Data Types

The documentation in this section has been obtained directly from the NXDL Schema file: *nxdl.xsd*. First, the basic elements are defined in alphabetical order. Attributes to an element are indicated immediately following the element and are preceded with an "@" symbol, such as **@attribute**. Then, the common data types used within the NXDL specification are defined. Pay particular attention to the rules for *validItemName* and *validNXClassName*.

NXDL Elements

attribute An `attribute` element can *only* be a child of a `field` or `group` element. It is used to define *attribute* elements to be used and their data types and possibly an enumeration of allowed values.

For more details, see: *attributeType*

definition A `definition` element can *only* be used at the root level of an NXDL specification. Note: Due to the large number of attributes of the `definition` element, they have been omitted from the figure below.

For more details, see: *definition*, *definitionType*, and *definitionTypeAttr*

dimensions The `dimensions` element describes the *shape* of an array. It is used *only* as a child of a `field` element.

For more details, see: *dimensionsType*

doc A `doc` element can be a child of most NXDL elements. In most cases, the content of the `doc` element will also become part of the NeXus manual.

element {any}:

In documentation, it may be useful to use an element that is not directly specified by the NXDL language. The *any* element here says that one can use any element at all in a `doc` element and NXDL will not process it but pass it through.

For more details, see: *docType*

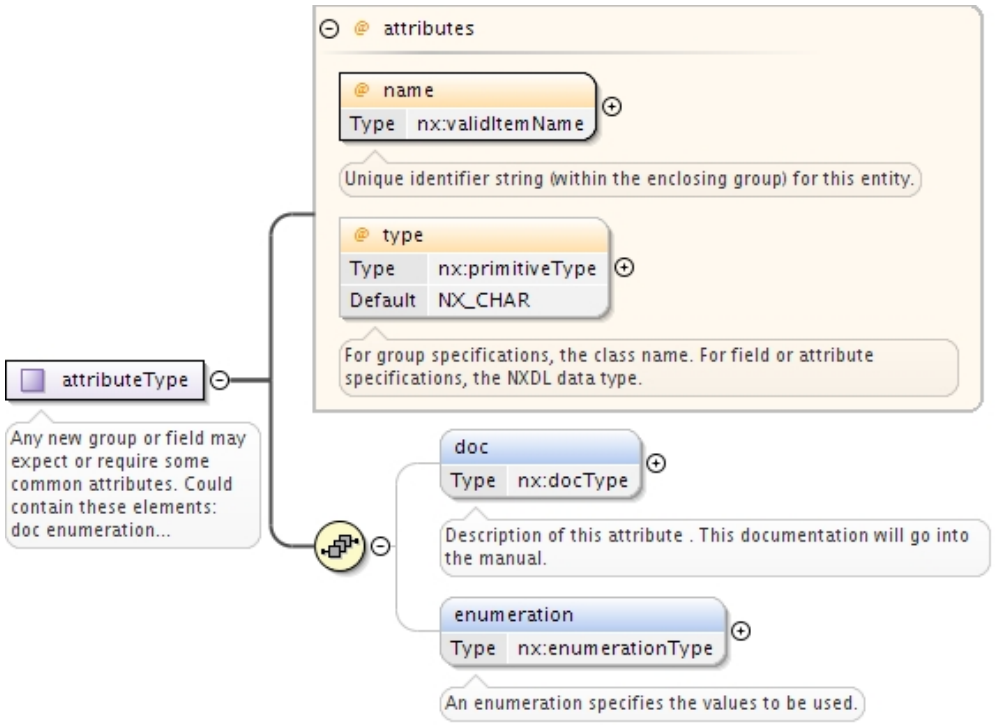


Figure 2.1: Graphical representation of the NXDL `attribute` element

enumeration An `enumeration` element can *only* be a child of a `field` or `attribute` element. It is used to restrict the available choices to a predefined list, such as to control varieties in spelling of a controversial word (such as *metre* vs. *meter*).

For more details, see: *enumerationType*

field The `field` element provides the value of a named item. Many different attributes are available to further define the `field`. Some of the attributes are not allowed to be used together (such as `axes` and `axis`); see the documentation of each for details. It is used *only* as a child of a `group` element.

For more details, see: *fieldType*

group A `group` element can *only* be a child of a `definition` or `group` element. It describes a common level of organization in a NeXus data file, similar to a subdirectory in a file directory tree.

For more details, see: *groupType*

link A `link` element can *only* be a child of a `field` or `group` element. It describes the path to the original source of the parent `field` or `group`.

For more details, see: *linkType*

symbols A `symbols` element can *only* be a child of a `definition` element. It defines the array index symbols to be used when defining arrays as `field` elements with common dimensions and lengths.

For more details, see: *symbolsType*

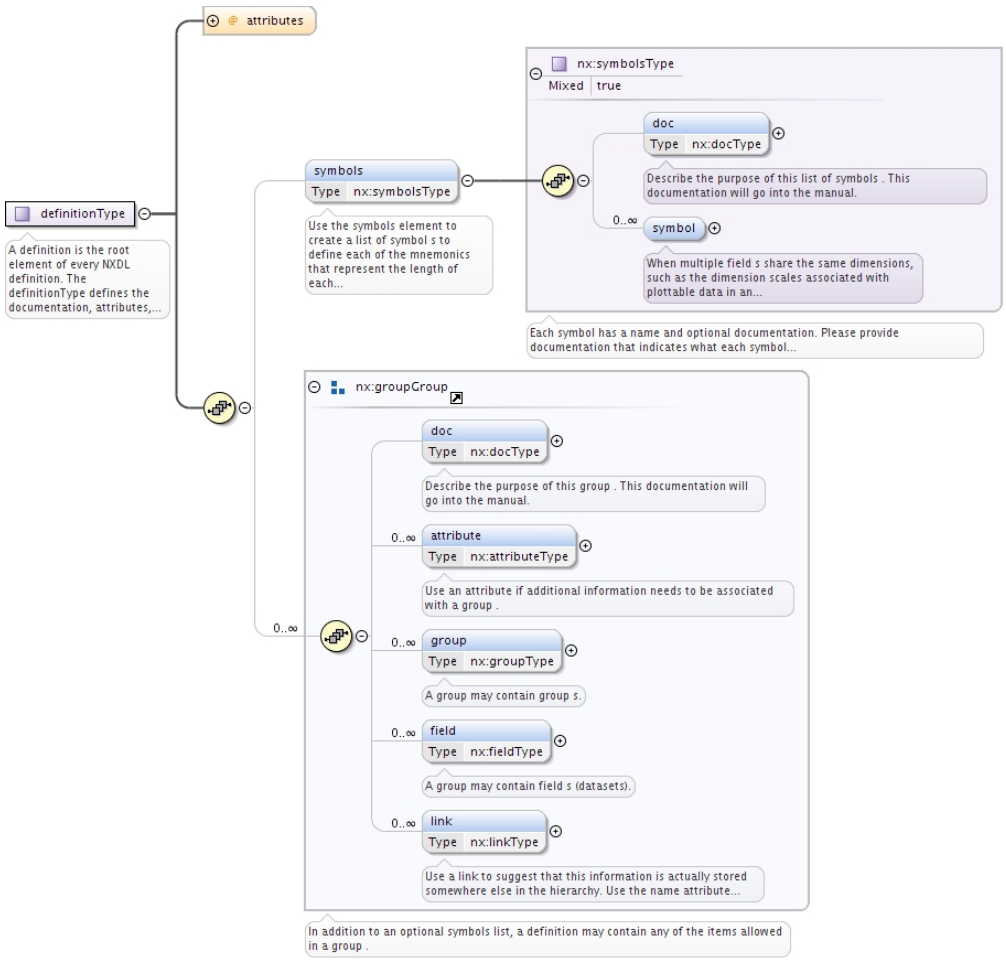


Figure 2.2: Graphical representation of the NXDL `definition` element

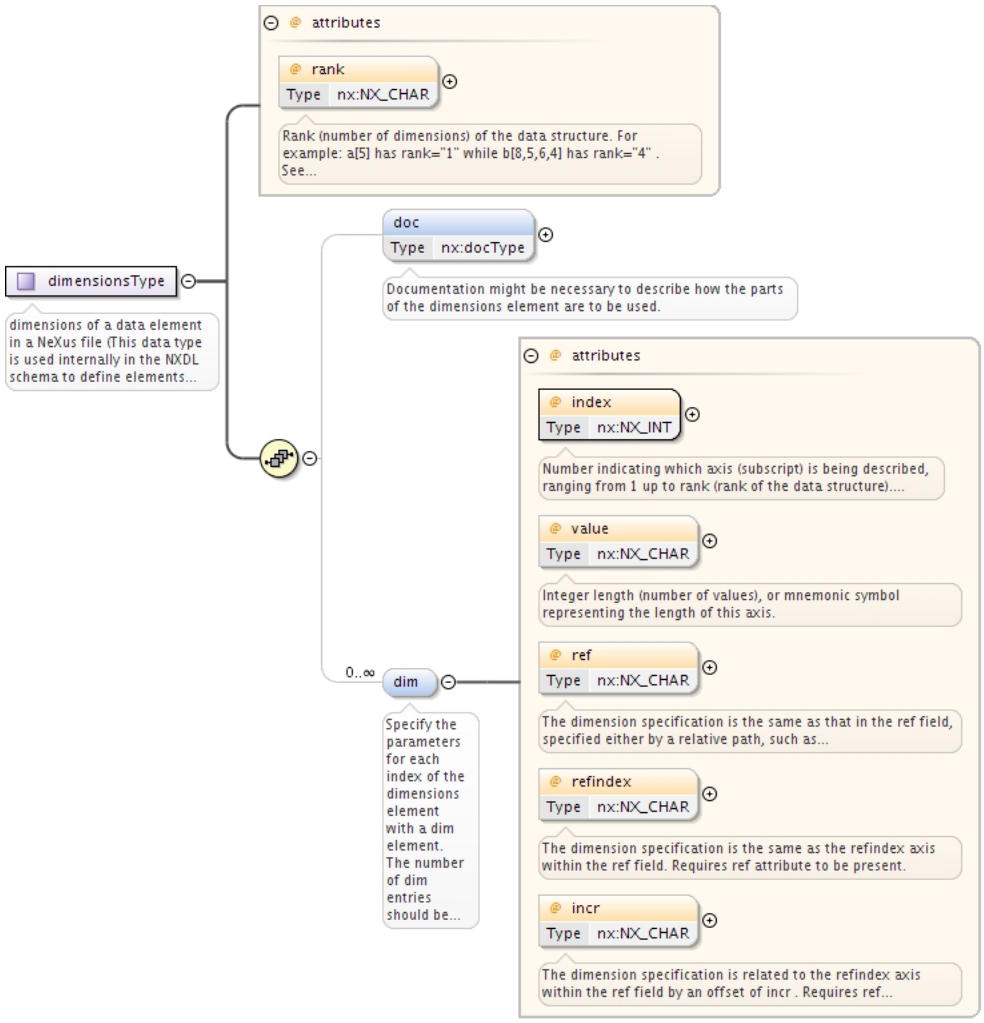


Figure 2.3: Graphical representation of the NXDL `dimensions` element

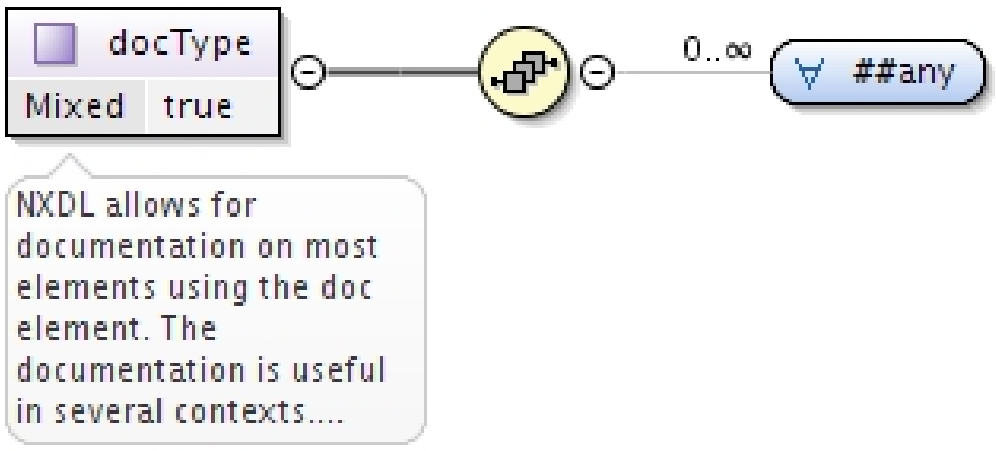


Figure 2.4: Graphical representation of the NXDL `doc` element

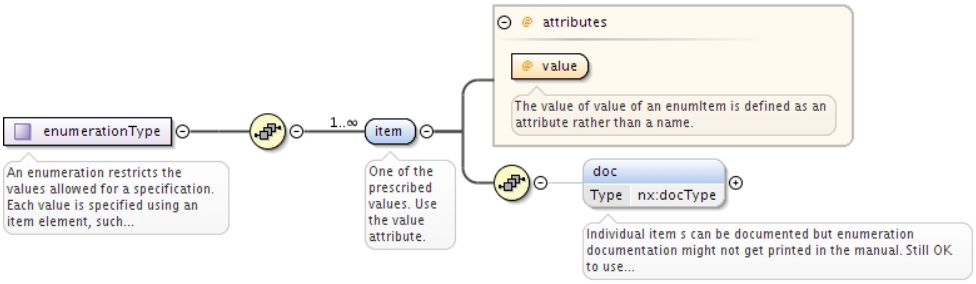


Figure 2.5: Graphical representation of the NXDL `enumeration` element

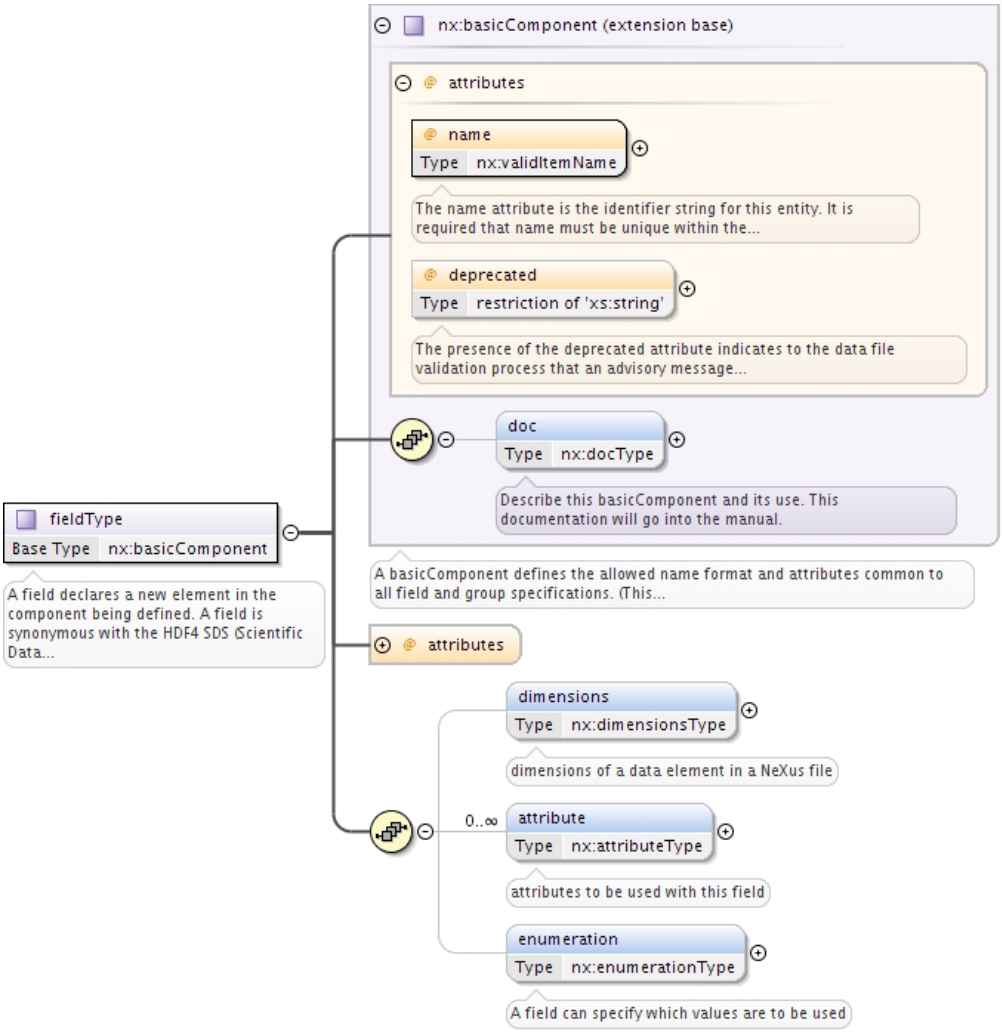


Figure 2.6: Graphical representation of the NXDL `field` element

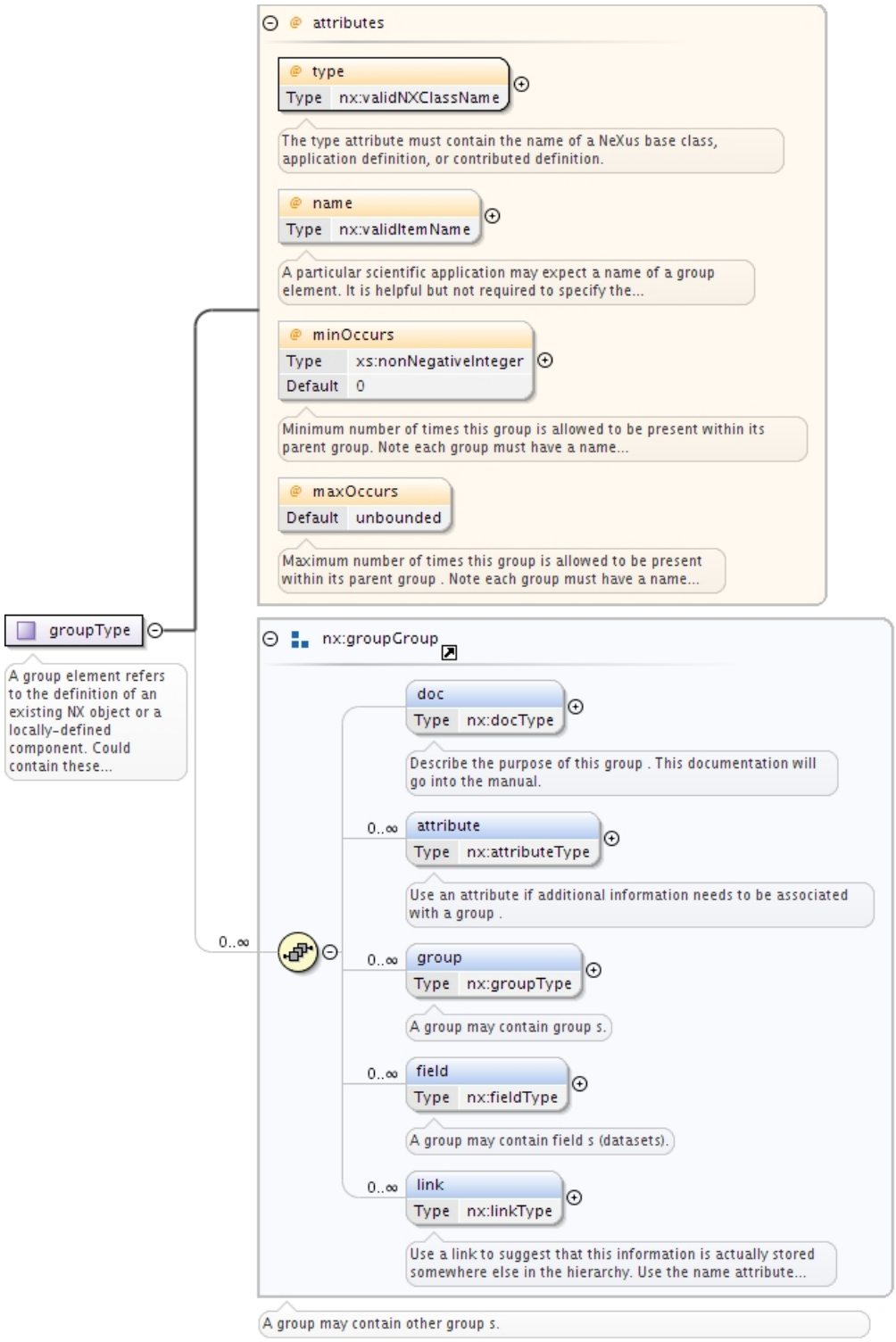


Figure 2.7: Graphical representation of the NXDL `group` element

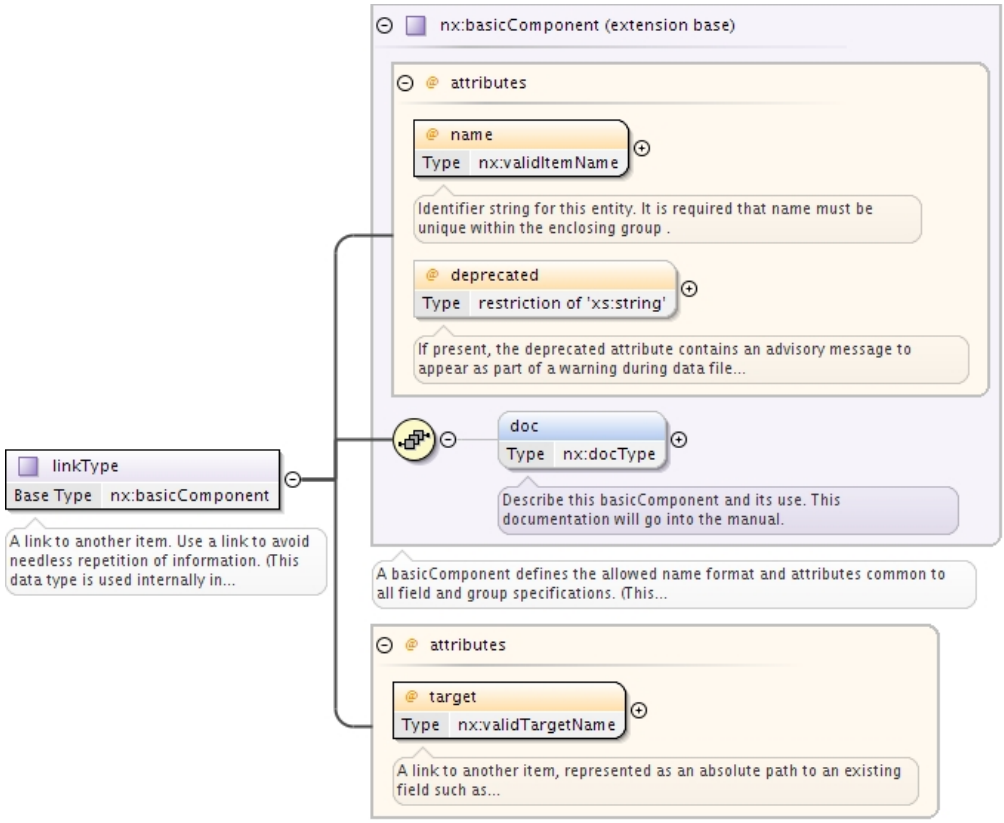


Figure 2.8: Graphical representation of the NXDL `link` element

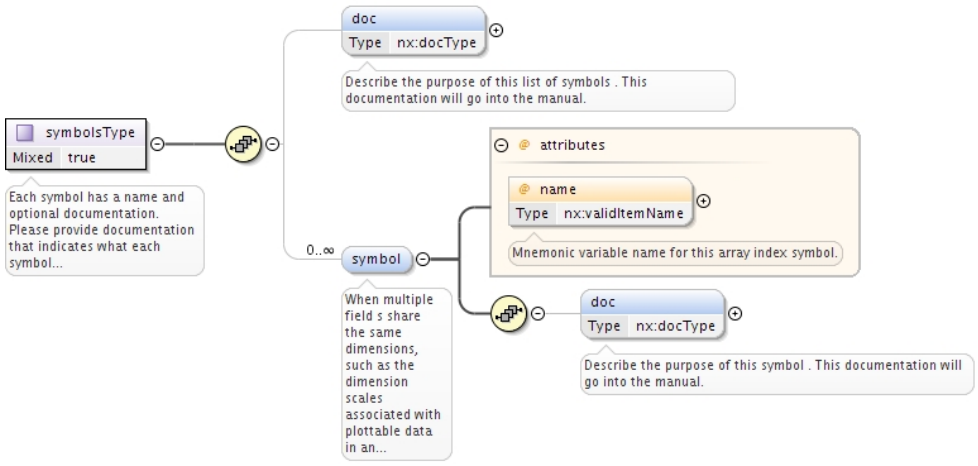


Figure 2.9: Graphical representation of the NXDL `symbols` element

NXDL Data Types (internal)

Data types that define the NXDL language are described here. These data types are defined in the XSD Schema (`nxdl.xsd`) and are used in various parts of the Schema to define common structures or to simplify a complicated entry. While the data types are not intended for use in NXDL specifications, they define structures that may be used in NXDL specifications.

attributeType Any new group or field may expect or require some common attributes.

(This data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

Attributes of attributeType

@name

Name of the attribute (unique within the enclosing group).

@type

Type of the attribute. For `group` specifications, the class name. For `field` or `attribute` specifications, the NXDL data type.

Elements of attributeType

doc

Description of this `attribute`. This documentation will go into the manual.

enumeration

An enumeration specifies the values to be used.

definition A `definition` element is the `group` at the root of every NXDL specification. It may *only* appear at the root of an NXDL file and must only appear **once** for the NXDL to be *well-formed*.

definitionType A `definition` is the root element of every NXDL definition. It may *only* appear at the root of an NXDL file and must only appear **once** for the NXDL to be *well-formed*.

The `definitionType` defines the documentation, attributes, fields, and groups that will be used as children of the `definition` element. Could contain these elements:

- `attribute`
- `doc`
- `field`
- `group`
- `link`

Note that a `definition` element also includes the definitions of the `basicComponent` data type. (The `definitionType` data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

Attributes of definitionType

@category

NXDL `base` definitions define the dictionary of terms to use for these components. All terms in a `base` definition are optional. NXDL `application` definitions define what is required for a scientific interest. All terms in an `application` definition are required. NXDL `contributed` definitions may be considered either base or applications. Contributed definitions <emphasis>must</emphasis> indicate their intended use, either as a base class or as an application definition.

@extends

The `extends` attribute allows this definition to *subclass* from another NXDL, otherwise `extends="NXobject"` should be used.

@ignoreExtraAttributes

Only validate known attributes; do not not warn about unknowns. The `ignoreExtraAttributes` attribute is a flag to the process of validating NeXus data files. By setting `ignoreExtraAttributes="true"`, presence of any undefined attributes in this class will not generate warnings during validation. Normally, validation will check all the attributes against their definition in the NeXus base classes and application definitions. Any items found that do not match the definition in the NXDL will generate a warning message.

The `ignoreExtraAttributes` attribute should be used sparingly!

@ignoreExtraFields

Only validate known fields; do not not warn about unknowns. The `ignoreExtraFields` attribute is a flag to the process of validating NeXus data files. By setting `ignoreExtraFields="true"`, presence of any undefined fields in this class will not generate warnings during validation. Normally, validation will check all the fields against their definition in the NeXus base classes and application definitions. Any items found that do not match the definition in the NXDL will generate a warning message.

The `ignoreExtraFields` attribute should be used sparingly!

@ignoreExtraGroups

Only validate known groups; do not not warn about unknowns. The `ignoreExtraGroups` attribute is a flag to the process of validating NeXus data files. By setting `ignoreExtraGroups="true"`, presence of any undefined groups in this class will not generate warnings during validation. Normally, validation will check all the groups against their definition in the NeXus base classes and application definitions. Any items found that do not match the definition in the NXDL will generate a warning message.

The `ignoreExtraGroups` attribute should be used sparingly!

@name

The `name` of this NXDL file (without the file extensions). The name must be unique amongst all the NeXus base class, application, and contributed definitions. For the class to be adopted by the NIAC, the first two letters must be "`NX`" (in uppercase). Any other use must *not* begin with "`NX`" in any combination of upper or lower case.

@restricts

The `restricts` attribute is a flag to the data validation. When `restricts="1"`, any non-standard component found (and checked for validity aginst this NXDL specification) in a NeXus data file will be flagged as an error. If the `restricts` attribute is not present, any such situations will produce a warning.

@svnid

The identifier string from the subversion revision control system. This reports the time stamp and the revision number of this file. (Updated automatically, unlike the `version` attribute.)

@type

Must be `type="group"`

@version

Version of *this* NXDL definition. Each NXDL specification may have a different version to facilitate software maintenance. This value is modified by the person who edits this file when this NXDL specification has changed significantly (in a way that downstream software should be aware).

Elements of definitionType

symbols

Use a `symbols` list to define each of the mnemonics that represent the length of each dimension in a vector or array.

Groups under definitionType

In addition to an optional `symbols` list, a `definition` may contain any of the items allowed in a `group`.

definitionTypeAttr Prescribes the allowed values for `definition type` attribute. (This data type is used internally in the NXDL schema to define a data type.)

The value may be any one from this list only:

- `group`
- `definition`

dimensionsType dimensions of a data element in a NeXus file (This data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

Attributes of dimensionsType

@rank

Rank (number of dimensions) of the data structure. For example: `a[5]` has `rank="1"` while `b[8,5,6,4]` has `rank="4"`. See http://en.wikipedia.org/wiki/Rank_(computer_programming) for more details.

Elements of dimensionsType

dim

Specify the parameters for each index of the `dimensions` element with a `dim` element. The number of `dim` entries should be equal to the `rank` of the array. For example, these terms describe a 2-D array with lengths (`nsurf`, `nwl`):

```
<dimensions rank="2">
    <dim index="1" value="nsurf"/>
    <dim index="2" value="nwl"/>
</dimensions>
```

The `value` attribute is used by NXDL and also by the NeXus data file validation tools to associate and coordinate the same array length across multiple fields in a group.

@incr

The dimension specification is related to the `refindex` axis within the `ref` field by an offset of `incr`. Requires `ref` and `refindex` attributes to be present.

@index

Number or symbol indicating which axis (subscript) is being described, ranging from 1 up to `rank` (rank of the data structure). For example, given an array `A[i,j,k]`, `index="1"` would refer to the `i` axis (subscript). (`NXdata` uses `index="0"` to indicate a situation when the specific index is not known *a priori*.)

@ref

The dimension specification is the same as that in the `ref` field, specified either by a relative path, such as `polar_angle` or `../Qvec` or absolute path, such as `/entry/path/to/follow/to/ref/field`.

@refindex

The dimension specification is the same as the `refindex` axis within the `ref` field. Requires `ref` attribute to be present.

@value

Integer length (number of values), or mnemonic symbol representing the length of this axis.

doc

Documentation might be necessary to describe how the parts of the `dimensions` element are to be used.

docType NXDL allows for documentation on most elements using the `doc` element. The documentation is useful in several contexts. The documentation will be rendered in the manual. Documentation, is provided as tooltips by some XML editors when editing NXDL files. Simple documentation can be typed directly in the NXDL:

```
<field name="name">
        <doc>Descriptive name of sample</doc>
</field>
```

This is suitable for basic descriptions that do not need extra formatting such as a bullet-list or a table. For more advanced control, use the rules of restructured text, such as in the *NXdetector* specification. Refer to examples in the NeXus base class NXDL files such as *NXdata*.

Could contain these elements:

- *any*

(This data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

enumerationType An `enumeration` restricts the values allowed for a specification. Each value is specified using an `item` element, such as: `<item value="Synchrotron X-ray Source"/>`. Could contain these elements:

- `doc`
- `item`

(This data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

```
<field name="mode">
        <doc>source operating mode</doc>
        <enumeration>
                <item value="Single Bunch"><doc>for storage rings</doc></item>
                <item value="Multi Bunch"><doc>for storage rings</doc></item>
                <!-- other sources could add to this -->
        </enumeration>
</field>
```

Elements of enumerationType

item

One of the prescribed values. Use the `value` attribute.

@value

The value of `value` of an `enumItem` is defined as an attribute rather than a name.

doc

Individual items can be documented but this documentation might not be printed in the *NeXus Reference Guide*.

fieldType A `field` declares a new element in the component being defined. A `field` is synonymous with the HDF4 SDS (Scientific Data Set) and the HDF5 *dataset* terms. Could contain these elements:

- `attribute`
- `dimensions`
- `doc`
- `enumeration`

Note that a `field` element also includes the definitions of the `basicComponent` data type. (The `fieldType` data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

@axes

Presence of the `axes` attribute means this field is an ordinate.

This attribute contains a colon (or comma in legacy files) delimited list of the names of independent axes when plotting this field. Each name in this list must exist as a field in the same group. <!– perhaps even discourage use of square brackets in axes attribute? –> (Optionally, the list can be enclosed by square brackets but this is not common.) The regular expression for this rule is:

```
[A-Za-z_][\w_]*([ :][A-Za-z_][\w_]*)*
```

@axis

NOTE: Use of this attribute is discouraged. It is for legacy support. You should use the axes attribute instead.

Presence of the `axis` attribute means this field is an abcissa.

The attribute value is an integer indicating this field as an axis that is part of the data set. The data set is a field with the attribute `signal=1` in the same group. The value can range from 1 up to the number of independent axes (abcissae) in the data set.

A value of `axis=1`" indicates that this field contains the data for the first independent axis. For example, the X axis in an XY data set.

A value of `axis=2` indicates that this field contains the data for the second independent axis. For example, the Y axis in a 2-D data set.

A value of `axis=3` indicates that this field contains the data for the third independent axis. For example, the Z axis in a 3-D data set.

A field with an `axis` attribute should not have a `signal` attribute.

@interpretation

This instructs the consumer of the data what the last dimensions of the data are. It allows plotting software to work out the natural way of displaying the data.

For example a single-element, energy-resolving, fluorescence detector with 512 bins should have `interpretation="spectrum"`. If the detector is scanned over a 512 x 512 spatial grid, the data reported will be of dimensions: 512 x 512 x 512. In this example, the initial plotting representation should default to data of the same dimensions of a 512 x 512 pixel `image` detector where the images where taken at 512 different pressure values.

In simple terms, the allowed values mean:

- `scaler` = 0-D data to be plotted
- `spectrum` = 1-D data to be plotted
- `image` = 2-D data to be plotted
- `vertex` = 3-D data to be plotted

@long_name

Descriptive name for this field (may include whitespace and engineering units). Often, the long_name (when defined) will be used as the axis label on a plot.

@maxOccurs

Defines the maximum number of times this element may be used. Its value is confined to zero or greater. Must be greater than or equal to the value for the "minOccurs" attribute. A value of "unbounded" is allowed.

@minOccurs

Defines the minimum number of times this element may be used. Its value is confined to zero or greater. Must be less than or equal to the value for the "maxOccurs" attribute. A value of "unbounded" is allowed.

@offset

The `stride` and `offset` attributes are used together to index the array of data items in a multi-dimensional array. They may be used as an alternative method to address a data array that is not stored in the standard NeXus method of "C" order.

The `offset` attribute determines the starting coordinates of the data array for each dimension.

See http://davis.lbl.gov/Manuals/HDF5-1.4.3/Tutor/phypereg.html or *4. Dataspace Selection Operations* in http://www.hdfgroup.org/HDF5/doc1.6/Dataspaces.html.

The `offset` attribute contains a comma-separated list of integers. (In addition to the required comma delimiter, whitespace is also allowed to improve readability.) The number of items in the list is equal to the rank of the data being stored. The value of each item is the offset in the array of the first data item of that subscript of the array.

@primary

Integer indicating the priority of selection of this field for plotting (or visualization) as an axis.

Presence of the `primary` attribute means this field is an abcissa.

@signal

Presence of the `signal` attribute means this field is an ordinate.

Integer marking this field as plottable data (ordinates). The value indicates the priority of selection or interest. Some facilities only use `signal=1` while others use `signal=2` to indicate plottable data of secondary interest. Higher numbers are possible but not common and interpretation is not standard.

A field with a `signal` attribute should not have an `axis` attribute.

@stride

The `stride` and `offset` attributes are used together to index the array of data items in a multi-dimensional array. They may be used as an alternative method to address a data array that is not stored in the standard NeXus method of "C" order.

The `stride` list chooses array locations from the data array with each value in the `stride` list determining how many elements to move in each dimension. Setting a value in the `stride` array to 1 moves to each element in that dimension of the data array, while setting a value of 2 in a location in the `stride` array moves to every other element in that dimension of the data array. A value in the `stride` list may be positive to move forward or negative to step backward. A value of zero will not step (and is of no particular use).

See http://davis.lbl.gov/Manuals/HDF5-1.4.3/Tutor/phypereg.html or *4. Dataspace Selection Operations* in http://www.hdfgroup.org/HDF5/doc1.6/Dataspaces.html.

The `stride` attribute contains a comma-separated list of integers. (In addition to the required comma delimiter, whitespace is also allowed to improve readability.) The number of items in the list is equal to the rank of the data being stored. The value of each item is the spacing of the data items in that subscript of the array.

@type

Defines the type of the element as allowed by the NAPI (NeXus Application Programmer Interface). See elsewhere for the complete list of allowed NAPI types.

@units

String describing the engineering units. The string should be appropriate for the value and should conform to the NeXus rules for units. Conformance is not validated at this time.

attribute

attributes to be used with this field

dimensions

dimensions of a data element in a NeXus file

enumeration

A field can specify which values are to be used

groupType A group element refers to the definition of an existing NX object or a locally-defined component. Could contain these elements:

- `attribute`
- `doc`
- `field`
- `group`
- `link`

Note that a group element also includes the definitions of the basicComponent data type. (The groupType data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

Attributes of groupType

@maxOccurs

Maximum number of times this group is allowed to be present within its parent group. Note each group must have a name attribute that is unique among all group and field declarations within a common parent group.

@minOccurs

Minimum number of times this group is allowed to be present within its parent group. Note each group must have a name attribute that is unique among all group and field declarations within a common parent group.

@name

A particular scientific application may expect a name of a group element. It is helpful but not required to specify the name attribute in the NXDL file. It is suggested to always specify a name to avoid ambiguity. It is also suggested to derive the name from the type, using an additional number suffix as necessary. For example, consider a data file with only one NXentry. The suggested default name would be entry. For a data file with two or more NXentry groups, the suggested names would be entry1, entry2, ... Alternatively, a scientific application such as small-angle scattering might require a different naming procedure; two different NXaperture groups might be given the names beam-defining slit and scatter slit.

@type

The type attribute *must* contain the name of a NeXus base class, application definition, or contributed definition.

linkType A link to another item. Use a link to avoid needless repetition of information. (This data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

@target

A link to another item, represented as an absolute path to an existing field such as /NXentry/NXinstrument/NXdetector/polar_angle. Could contain these elements:

- doc

Matching regular expression:

```
(/[a-zA-Z_][\w_]*(:[a-zA-Z_][\w_]*)?)+
```

symbolsType Each symbol has a name and optional documentation. Please provide documentation that indicates what each symbol represents. For example:

```
<symbols>
    <symbol name="nsurf"><doc>number of reflecting surfaces</doc></symbol>
    <symbol name="nwl"><doc>number of wavelengths</doc></symbol>
</symbols>
```

Elements of symbolsType

doc

Describe the purpose of this list of `symbols`. This documentation will go into the manual.

symbol

When multiple `field` elements share the same dimensions, such as the dimension scales associated with plottable data in an `NXdata` group, the length of each dimension written in a NeXus data file should be something that can be tested by the data file validation process.

@name

Mnemonic variable name for this array index symbol.

doc

Describe the purpose of the parent `symbol`. This documentation will go into the manual.

basicComponent A `basicComponent` defines the allowed name format and attributes common to all `field` and `group` specifications. (This data type is used internally in the NXDL schema to define elements and attributes to be used by users in NXDL specifications.)

Attributes of basicComponent

@deprecated

The presence of the `deprecated` attribute indicates to the data file validation process that an advisory message (specified as the content of the `deprecated` attribute) will be reported. Future versions of the NXDL file might not define (or even re-use) this component. For example:

```
deprecated="as of release MAJOR.MINOR"
```

Note: because `deprecated` is an attribute, the XML rules do not permit it to have any element content.

@name

The `name` attribute is the identifier string for this entity. It is required that `name` must be unique within the enclosing `group`. The rule (`validItemName`) is defined to only allow names that can be represented as valid variable names in most computer languages.

Elements of basicComponent

doc

Describe this `basicComponent` and its use. This documentation will go into the manual.

validItemName Used for allowed names of elements and attributes. Need to be restricted to valid program variable names. Note: This means no "-" or "." characters can be allowed and you cannot start with a number. HDF4 had a 64 character limit on names (possibly including NULL) and NeXus enforces this via the `NX_MAXNAMELEN` variable with a **64** character limit (which may be 63 on a practical basis if one considers a NULL terminating byte). (This data type is used internally in the NXDL schema to define a data type.)

The value may be any `xs:token` that *also* matches the regular expression:

```
[A-Za-z_][\w_]*
```

validNXClassName Used for allowed names of NX class types (e.g. NXdetector) not the instance (e.g. bank1) which is covered by validItemName. (This data type is used internally in the NXDL schema to define a data type.)

The value may be any `nx:validItemName` that *also* matches the regular expression:

```
NX.+
```

validTargetName This is a valid link target - currently it must be an absolute path made up of valid names with the / character delimiter. But we may want to consider allowing "`..`" (parent of directory) at some point. If the `name` attribute is helpful, then use it in the path with the syntax of *name:type* as in these examples:

```
/NXentry/NXinstrument/analyzer:NXcrystal/ef
/NXentry/NXinstrument/monochromator:NXcrystal/ei
/NX_other
```

Must also consider use of `name` attribute in resolving `link` targets. (This data type is used internally in the NXDL schema to define a data type.)

From the HDF5 documentation (http://www.hdfgroup.org/HDF5/doc/UG/UG_frame09Groups.html):

> *Note that relative path names in HDF5 do not employ the ''../'' notation, the UNIX notation indicating a parent directory, to indicate a parent group.*

Thus, if we only consider the case of `[name:]type`, the matching regular expression syntax is written: `/[a-zA-Z_][\w_]*(:[a-zA-Z_][\w_]*)?)+`. Note that HDF5 also permits relative path names, such as: `GroupA/GroupB/Dataset1` but this is not permitted in the matching regular expression and not supported in NAPI.

The value may be any `xs:token` that *also* matches the regular expression:

```
(/[a-zA-Z_][\w_]*(:[a-zA-Z_][\w_]*)?)+
```

nonNegativeUnbounded A `nonNegativeUnbounded` allows values including all positive integers, zero, and the string `unbounded`. (This data type is used internally in the NXDL schema to define a data type.)

The `xs:string` data type The `xs:string` data type can contain characters, line feeds, carriage returns, and tab characters. See http://www.w3schools.com/Schema/schema_dtypes_string.asp for more details.

The `xs:token` data type The `xs:string` data type is derived from the `xs:string` data type.

The `xs:token` data type also contains characters, but the XML processor will remove line feeds, carriage returns, tabs, leading and trailing spaces, and multiple spaces. See http://www.w3schools.com/Schema/schema_dtypes_string.asp for more details.

NXDL: Data Types and Units

Data Types allowed in NXDL specifications

Data types for use in NXDL describe the expected type of data for a NeXus field. These terms are very broad. More specific terms are used in actual NeXus data files that describe size and array dimensions. In addition to the types in the following table, the `NAPI` type is defined when one wishes to permit a field with any of these data types.

ISO8601 ISO 8601 date and time representation (http://www.w3.org/TR/NOTE-datetime)

NX_BINARY any representation of binary data - if text, line terminator is [CR][LF]

NX_BOOLEAN true/false value (true | 1 | false | 0)

NX_CHAR any string representation

NX_DATE_TIME alias of ISO8601

NX_FLOAT any representation of a floating point number

NX_INT any representation of an integer number

NX_NUMBER any valid NeXus number representation

NX_POSINT any representation of a positive integer number (greater than zero)

NX_UINT any representation of an unsigned integer number (includes zero)

Unit Categories allowed in NXDL specifications

Unit categories in NXDL specifications describe the expected type of units for a NeXus field. They should describe valid units consistent with the *NeXus units* section. The values for unit categories are restricted (by an enumeration) to the following table.

NX_ANGLE example: degrees or radians or arcminutes or

NX_ANY usage: things like logs that aren't picky on units

NX_AREA example: m2 or barns

NX_CHARGE example: pC or C

NX_CROSS_SECTION example: barns

NX_CURRENT example: A

NX_DIMENSIONLESS for fields where the units cancel out, example: "" or mm/mm (NOTE: not the same as NX_UNITLESS)

NX_EMITTANCE emittance (`length * angle`) of a radiation source, example: `nm*rad`

NX_ENERGY example: J or keV

NX_FLUX example: s-1 cm-2

NX_FREQUENCY example: Hz

NX_LENGTH example: m

NX_MASS example: g

NX_MASS_DENSITY example: g cm-3

NX_MOLECULAR_WEIGHT example: g mol-1

NX_PERIOD (alias to NX_TIME) period of pulsed source, example: microseconds

NX_PER_AREA example: cm-2

NX_PER_LENGTH example: cm-1

NX_POWER example: W

NX_PRESSURE example: Pa

NX_PULSES (alias to NX_NUMBER) clock pulses

NX_SCATTERING_LENGTH_DENSITY example: cm-2

NX_SOLID_ANGLE example: sr | steradian

NX_TEMPERATURE example: K

NX_TIME example: s

NX_TIME_OF_FLIGHT (alias to NX_TIME) example: s

NX_UNITLESS for fields that don't have a unit (e.g. hkl) so that they don't inherit the wrong units (NOTE: not the same as NX_DIMENSIONLESS)

NX_VOLTAGE example: V

NX_VOLUME example: m3

NX_WAVELENGTH example: Angstrom

NX_WAVENUMBER units for Q, example: Angstrom-1 or nm-1

2.3 NeXus class definitions

Information is stored in a NeXus data file by grouping together similar parts. For example, information about the sample could include a descriptive name, the temperature, and other items. NeXus specifies the contents of these groupings using *classes*. In some parts of this manual, these classes might be called *group type* or some similar term. In this section, the NeXus classes are described in detail. Each class is specified using *NXDL: The NeXus Definition Language*, described in a separate chapter.

There are three types of NeXus class file: base classes, application definitions, and contributed definitions. Base class definitions define the *complete* set of terms that *might* be used in an instance of that class. Application definitions define the *minimum* set of terms that *must* be used in an instance of that class. Contributed definitions include propositions from the community for NeXus base classes or application definitions, as well as other NXDL files for long-term archival by NeXus.

2.3.1 NeXus Class Specifications

Overview of NeXus classes

Each of the NeXus classes is described in two basic ways. First, a short list of descriptive information is provided as a header, then a condensed listing of the basic structure, then a table providing documentation for the various components of the NeXus class.

category The category of NXDL, one of these:

- `base` (meaning: *base class*)
- `application` (meaning: *application definition*)
- `contributed` (meaning: *contributed definition*)

NXDL source Name of the NeXus class and a URL to the source listing in the NeXus subversion repository.

version A string that documents this particular version of this NXDL.

SVN Id Subversion repository checkout identification, stripped of the surrounding dollar signs. (The *Id* is blank on files copied direct from the repository that are not checked out by a subversion client.)

NeXus Definition Language *NXDL: The NeXus Definition Language* is used to describe the components in the NeXus Base Classes, as well as application and contributed definitions. The intent of NXDL is to provide a rules-based method for defining a NeXus data file that is specific to either an instrument (where NeXus has been for years) or an area of scientific technique or analysis. NXDL replaces the meta-DTD method used previously to define the NeXus base classes.

extends class NeXus class extended by this class. Most NeXus base classes only extend the base class definition (NXDL).

other classes included List (including URLs) of other classes used to define this class.

symbol list List of the `symbols` (if present) that define mnemonics that represent the length of each dimension in a vector or array.

documentation Description of the NeXus class. DocBook markup (formatting is allowed).

Basic structure of the class

A compact listing of the basic structure (groups, fields, dimensions, attributes, and links) is prepared for each NXDL specification. Indentation shows nested structure. Attributes are prepended with the `@` symbol while links use the characters `-->` to represent the path to the intended source of the information.

The table has columns to describe the basic information about each field or group in the class. An example of the varieties of specifications are given in the following table using items found in various NeXus base classes.

Name	Type	Units	Description (and Occurrences)
`program_name`	NX_CHAR		Name of program used to generate this file
`@version`	NX_CHAR		Program version number Occurences: 1 : *default*
`@configuration`	NX_CHAR		configuration of the program
`thumbnail`	*NXnote*		A small image that is representative of the entry. An example of this is a 640x480 JPEG image automatically produced by a low resolution plot of the NXdata.
`@mime_type`	NX_CHAR		expected: *mime_type="image/*"*
	NXgeometry		describe the geometry of this class
`distance`	NX_FLOAT	NX_LENGTH	Distance from sample
`mode`	"Single Bunch" | "Multi Bunch"		source operating mode
`target_material`	Ta | W | depleted_U | enriched_U | Hg | Pb | C		Pulsed source target material

In the above example, the fields might appear in a NeXus XML data file as

Example fragment of a NeXus XML data file

```
    <program_name version="1.0a" configuration="standard">
        MaxSAS
    </program_name>
    <NXnote name="thumbnail" mime_type="image/*">
        <!-- contents of an NXnote would appear here -->
    </NXnote>
    <distance units="mm">125.6</distance>
    <mode> Single Bunch </mode>
    <target_material>depleted_U</target_material>
```

The columns in the table are described as follows:

Name (and attributes) Name of the data field. Since `name` needs to be restricted to valid program variable names, no "–" characters can be allowed. Name must satisfy both HDF and XML naming.

```
NameStartChar ::=  _ | a..z | A..Z
NameChar      ::=  NameStartChar | 0..9
Name          ::=  NameStartChar (NameChar)*

Or, as a regular expression:     [_a-zA-Z][_a-zA-Z0-9]*
equivalent regular expression:   [_a-zA-Z][\w_]*
```

Attributes, identified with a leading "at" symbol (`@`) and belong with the preceding field or group, are additional metadata used to define this field or group. In the example above, the `program_name` element has two attributes: `version` (required) and `configuration` (optional) while the `thumbnail` element has one attribute: `mime_type` (optional).

Type Type of data to be represented by this variable. The type is one of those specified in *NXDL: The NeXus Definition Language*. In the case where the variable can take only one value from a known list, the list of known values is presented, such as in the `target_material` field above: `Ta | W | depleted_U | enriched_U | Hg | Pb | C`. Selections with included whitespace are surrounded by quotes. See the example above for usage.

Units Data units, given as character strings, must conform to the NeXus units standard. See the *NeXus units* section for details.

Description (and Occurrences) A simple text description of the data field. No markup or formatting is allowed. The absence of *Occurrences* in the item description signifies that both `minOccurs` and `maxOccurs` have the default values. If the number of occurrences of an item are specified in the NXDL (through `@minOccurs` and `@maxOccurs` attributes), they will be reported in the Description column similar to the example shown above. Default values for occurrences are shown in the following table. The `NXDL element type` is either a group (such as a NeXus base class), a field (that specifies the name and type of a variable), or an attribute of a field or group. The number of times an item can appear ranges between `minOccurs` and `maxOccurs`. A default `minOccurs` of zero means the item is optional. For attributes, `maxOccurs` cannot be greater than 1.

NXDL element type	minOccurs	maxOccurs
group	0	unbounded
field	0	unbounded
attribute	0	1

NeXus Class Specifications

Base Class Definitions

A description of each NeXus base class definition is given. NeXus base class definitions define the *complete* set of terms that *might* be used in an instance of that class. Consider the base classes as a set of *components* that are used to construct a data file.

NXaperture

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*, *NXnote*

Template of a beamline aperture.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXaperture.nxdl.xml

svnid: Id: NXaperture.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXaperture

```
NXaperture (base class, version 1.0)
  description:NX_CHAR
  material:NX_CHAR
  NXgeometry
  NXgeometry
  NXnote
```

NXaperture Members Declarations in the *NXaperture* group.

Name and Attributes	Type	Units	Description (and Occurrences)
material	(*NX_CHAR*)		Absorbing material of the aperture
description	(*NX_CHAR*)		Description of aperture
(geometry)	*NXgeometry*		location and shape of aperture
(geometry)	*NXgeometry*		location and shape of each blade
(note)	*NXnote*		describe an additional information in a note*

NXattenuator

version	category	extends	groups cited
1.0	base	*NXobject*	none

Description of a device that reduces the intensity of a beam by attenuation. If uncertain whether to use `NXfilter` (band-pass filter) or `NXattenuator` (reduces beam intensity), then choose `NXattenuator`.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXattenuator.nxdl.xml

svnid: Id: NXattenuator.nxdl.xml 1144 2012-09-21 11:13:24Z Pete Jemian

Structure of NXattenuator

```
NXattenuator (base class, version 1.0)
  absorption_cross_section:NX_FLOAT
  attenuator_transmission:NX_FLOAT
  distance:NX_FLOAT
  scattering_cross_section:NX_FLOAT
  status:NX_CHAR
    @time
  thickness:NX_FLOAT
  type:NX_CHAR
```

NXattenuator Members Declarations in the *NXattenuator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Distance from sample
type	(*NX_CHAR*)		Type or composition of attenuator, e.g. polythene
thickness	*NX_FLOAT*	*NX_LENGTH*	Thickness of attenuator along beam direction
scattering_cross_section	*NX_FLOAT*	*NX_CROSS_SECTION*	Scattering cross section (coherent+incoherent)
absorption_cross_section	*NX_FLOAT*	*NX_CROSS_SECTION*	Absorption cross section
attenuator_transmission	*NX_FLOAT*	*NX_DIMENSIONLESS*	The nominal amount of the beam that gets through (transmitted intensity)/(incident intensity)
status	(*NX_CHAR*)		In or out or moving of the beam Any of these value(s): • `in`: • `out`: • `moving`:
@time	*NX_DATE_TIME*		time stamp for this observation

NXbeam

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*

Template of the state of the neutron or X-ray beam at any location. It will be referenced by beamline component groups within the NXinstrument group or by the NXsample group. Note that variables such as the incident energy could be scalar values or arrays. This group is especially valuable in storing the results of instrument simulations in which it is useful to specify the beam profile, time distribution etc. at each beamline component. Otherwise, its most likely use is in the NXsample group in which it defines the results of the neutron scattering by the sample, e.g., energy transfer, polarizations.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXbeam.nxdl.xml

svnid: Id: NXbeam.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXbeam

```
NXbeam (base class, version 1.0)
  distance:NX_FLOAT
  energy_transfer:NX_FLOAT[i]
  final_beam_divergence:NX_FLOAT[2,j]
  final_energy:NX_FLOAT[i]
  final_polarization:NX_FLOAT[2,j]
```

```
  final_wavelength:NX_FLOAT[i]
  final_wavelength_spread:NX_FLOAT[i]
  flux:NX_FLOAT[i]
  incident_beam_divergence:NX_FLOAT[2,j]
  incident_energy:NX_FLOAT[i]
  incident_polarization:NX_FLOAT[2,j]
  incident_wavelength:NX_FLOAT[i]
  incident_wavelength_spread:NX_FLOAT[i]
  NXdata
```

NXbeam Members Declarations in the *NXbeam* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Distance from sample
incident_energy	*NX_FLOAT*	*NX_ENERGY*	Energy on entering beamline component Dimensions: (rank=1) • 1: `i`
final_energy	*NX_FLOAT*	*NX_ENERGY*	Energy on leaving beamline component Dimensions: (rank=1) • 1: `i`
energy_transfer	*NX_FLOAT*	*NX_ENERGY*	Energy change caused by beamline component Dimensions: (rank=1) • 1: `i`
incident_wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	Wavelength on entering beamline component Dimensions: (rank=1) • 1: `i`
incident_wavelength_spread	*NX_FLOAT*	*NX_WAVELENGTH*	Wavelength spread FWHM on entering component Dimensions: (rank=1) • 1: `i`
incident_beam_divergence	*NX_FLOAT*	*NX_ANGLE*	Divergence of beam entering this component Dimensions: (rank=2) • 1: `2` • 2: `j`
final_wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	Wavelength on leaving beamline component Dimensions: (rank=1) • 1: `i`
incident_polarization	*NX_FLOAT*	*NX_ANY*	Polarization vector on entering beamline component Dimensions: (rank=2) • 1: `2` • 2: `j`
final_polarization	*NX_FLOAT*	*NX_ANY*	Polarization vector on leaving beamline component Dimensions: (rank=2) • 1: `2` • 2: `j`
final_wavelength_spread	*NX_FLOAT*	*NX_WAVELENGTH*	Wavelength spread FWHM of beam leaving this component Dimensions: (rank=1) • 1: `i`
final_beam_divergence	*NX_FLOAT*	*NX_ANGLE*	Divergence FWHM of beam leaving this component Dimensions: (rank=2) • 1: `2` • 2: `j`
flux	*NX_FLOAT*	*NX_FLUX*	flux incident on beam plane area Dimensions: (rank=1) • 1: `i`
(data)	*NXdata*		Distribution of beam with respect to relevant

NXbeam_stop

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*

A class for a beamstop. Beamstops and their positions are important for SANS and SAXS experiments.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXbeam_stop.nxdl.xml

svnid: Id: NXbeam_stop.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXbeam_stop

```
NXbeam_stop (base class, version 1.0)
  description:NX_CHAR
  distance_to_detector:NX_FLOAT
  size:NX_FLOAT
  status:NX_CHAR
  x:NX_FLOAT
  y:NX_FLOAT
  NXgeometry
```

NXbeam_stop Members Declarations in the *NXbeam_stop* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		description of beamstop Any of these value(s): • `circular`: • `rectangular`:
size	*NX_FLOAT*	*NX_LENGTH*	size of beamstop
x	*NX_FLOAT*	*NX_LENGTH*	x position of the beamstop in relation to the detector
y	*NX_FLOAT*	*NX_LENGTH*	y position of the beamstop in relation to the detector
distance_to_detector	*NX_FLOAT*	*NX_LENGTH*	distance of the beamstop to the detector
status	(*NX_CHAR*)		Any of these value(s): • `in`: • `out`:
(geometry)	*NXgeometry*		engineering shape, orientation and position of the beam stop.

NXbending_magnet

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*, *NXgeometry*

description for a bending magnet

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXbending_magnet.nxdl.xml

svnid: Id: NXbending_magnet.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXbending_magnet

```
NXbending_magnet (base class, version 1.0)
  accepted_photon_beam_divergence:NX_FLOAT
  bending_radius:NX_FLOAT
  critical_energy:NX_FLOAT
  divergence_x_minus:NX_FLOAT
  divergence_x_plus:NX_FLOAT
  divergence_y_minus:NX_FLOAT
  divergence_y_plus:NX_FLOAT
  magnetic_field:NX_FLOAT
  source_distance_x:NX_FLOAT
  source_distance_y:NX_FLOAT
  spectrum:NXdata
  NXgeometry
```

NXbending_magnet Members Declarations in the *NXbending_magnet* group.

Name and Attributes	Type	Units	Description (and Occurrences)
critical_energy	*NX_FLOAT*	*NX_ENERGY*	
bending_radius	*NX_FLOAT*	*NX_LENGTH*	
magnetic_field	*NX_FLOAT*	*NX_CURRENT*	strength of magnetic field of dipole magnets
accepted_photon_beam_divergence	*NX_FLOAT*	*NX_LENGTH*	An array of four numbers giving X+, X-, Y+ and Y- half divergence
source_distance_x	*NX_FLOAT*	*NX_LENGTH*	Distance of source point from particle beam waist in X (horizontal) direction.
source_distance_y	*NX_FLOAT*	*NX_LENGTH*	Distance of source point from particle beam waist in Y (vertical) direction.
divergence_x_plus	*NX_FLOAT*	*NX_ANGLE*	Accepted photon beam divergence in X+ (horizontal outboard) direction. Note that divergence_x_plus+divergence_x_minus is the total horizontal beam divergence.
divergence_x_minus	*NX_FLOAT*	*NX_ANGLE*	Accepted photon beam divergence in X- (horizontal inboard) direction. Note that divergence_x_plus+divergence_x_minus is the total horizontal beam divergence.
divergence_y_plus	*NX_FLOAT*	*NX_ANGLE*	Accepted photon beam divergence in Y+ (vertical upward) direction. Note that divergence_y_plus+divergence_y_minus is the total vertical beam divergence.
divergence_y_minus	*NX_FLOAT*	*NX_ANGLE*	Accepted photon beam divergence in Y- (vertical downward) direction. Note that divergence_y_plus+divergence_y_minus is the total vertical beam divergence.
spectrum	*NXdata*		bending magnet spectrum
(geometry)	*NXgeometry*		"Engineering" position of bending magnet

NXcapillary

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*

This is a dictionary of field names to use for describing a capillary as used in X-ray beamlines. Based on information provided by Gerd Wellenreuther.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXcapillary.nxdl.xml

svnid: Id: NXcapillary.nxdl.xml 662 2010-10-21 15:47:35Z Pete Jemian

Structure of NXcapillary

```
NXcapillary (base class, version 1.0)
  accepting_aperture:NX_FLOAT
  focal_size:NX_FLOAT
  manufacturer:NX_CHAR
  maximum_incident_angle:NX_FLOAT
  type:NX_CHAR
  working_distance:NX_FLOAT
  gain:NXdata
  transmission:NXdata
```

NXcapillary Members Declarations in the *NXcapillary* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	*NX_CHAR*		Type of the capillary Any of these value(s): • `single_bounce`: • `polycapillary`: • `conical_capillary`:
manufacturer	*NX_CHAR*		The manufacturer of the capillary. This is actually important as it may have an impact on performance.
maximum_incident_angle	*NX_FLOAT*	*NX_ANGLE*	
accepting_aperture	*NX_FLOAT*	*NX_ANGLE*	
working_distance	*NX_FLOAT*	*NX_LENGTH*	
focal_size	*NX_FLOAT*		The focal size in FWHM
gain	*NXdata*		The gain of the capillary as a function of energy
transmission	*NXdata*		The transmission of the capillary as a function of energy

NXcharacterization

version	category	extends	groups cited
1.0	base	*NXobject*	none

note: This base class may be removed in future releases of NXDL. If you have a use for this base class, please provide a description of your intended use to the NIAC (nexus-committee@nexusformat.org).

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXcharacterization.nxdl.xml

svnid: Id: NXcharacterization.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXcharacterization

```
NXcharacterization (base class, version 1.0)
  @source
  @location
  @mime_type
```

```
  definition:NX_CHAR
    @version
    @URL
```

NXcharacterization Members Declarations in the *NXcharacterization* group.

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		
@version	(*NX_CHAR*)		
@URL	(*NX_CHAR*)		

NXcollection

version	category	extends	groups cited
1.0	contributed	*NXobject*	none

Use `NXcollection` to gather together any set of terms. The original suggestion is to use this as a container class for the description of a beamline.

For NeXus validation, `NXcollection` will always generate a warning since it is always an optional group. Anything (groups, fields, or attributes) placed in an `NXcollection` group will not be validated.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXcollection.nxdl.xml

svnid: Id: NXcollection.nxdl.xml 1209 2013-05-07 16:29:51Z Pete Jemian

Structure of NXcollection

```
NXcollection (contributed definition, version 1.0)
  beamline:NX_CHAR
```

NXcollection Members Declarations in the *NXcollection* group.

Name and Attributes	Type	Units	Description (and Occurrences)
beamline	(*NX_CHAR*)		name of the beamline for this collection

NXcollimator

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*, *NXlog*

Template of a beamline collimator.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXcollimator.nxdl.xml

svnid: Id: NXcollimator.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXcollimator

```
NXcollimator (base class, version 1.0)
  absorbing_material:NX_CHAR
  blade_spacing:NX_FLOAT
  blade_thickness:NX_FLOAT
  divergence_x:NX_FLOAT
```

```
  divergence_y:NX_FLOAT
  frequency:NX_FLOAT
  soller_angle:NX_FLOAT
  transmitting_material:NX_CHAR
  type:NX_CHAR
  NXgeometry
  frequency_log:NXlog
```

NXcollimator Members Declarations in the *NXcollimator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		Any of these value(s): • `Soller`: • `radial`: • `oscillating`: • `honeycomb`:
soller_angle	*NX_FLOAT*	*NX_ANGLE*	Angular divergence of Soller collimator
divergence_x	*NX_FLOAT*	*NX_ANGLE*	divergence of collimator in local x direction
divergence_y	*NX_FLOAT*	*NX_ANGLE*	divergence of collimator in local y direction
frequency	*NX_FLOAT*	*NX_FREQUENCY*	Frequency of oscillating collimator
blade_thickness	*NX_FLOAT*	*NX_LENGTH*	blade thickness
blade_spacing	*NX_FLOAT*	*NX_LENGTH*	blade spacing
absorbing_material	(*NX_CHAR*)		name of absorbing material
transmitting_material	(*NX_CHAR*)		name of transmitting material
(geometry)	*NXgeometry*		position, shape and size
frequency_log	*NXlog*		Log of frequency

NXcrystal

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*, *NXgeometry*, *NXlog*, *NXshape*

Template of a crystal monochromator or analyzer. Permits double bent monochromator comprised of multiple segments with anisotropic Gaussian mosaic.

If curvatures are set to zero or are absent, array is considered to be flat.

Scattering vector is perpendicular to surface. Crystal is oriented parallel to beam incident on crystal before rotation, and lies in vertical plane.

symbol list: These symbols will be used below to coordinate dimensions with the same lengths.

n_comp number of different unit cells to be described

i number of wavelengths

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXcrystal.nxdl.xml

svnid: Id: NXcrystal.nxdl.xml 1161 2012-09-26 17:24:04Z Pete Jemian

Structure of NXcrystal

```
NXcrystal (base class, version 1.0)
  azimuthal_angle:NX_FLOAT[i]
  bragg_angle:NX_FLOAT[i]
  chemical_formula:NX_CHAR
  curvature_horizontal:NX_FLOAT
```

```
  curvature_vertical:NX_FLOAT
  cut_angle:NX_FLOAT
  cylindrical_orientation_angle:NX_NUMBER
  d_spacing:NX_FLOAT
  density:NX_NUMBER
  is_cylindrical:NX_BOOLEAN
  mosaic_horizontal:NX_FLOAT
  mosaic_vertical:NX_FLOAT
  order_no:NX_INT
  orientation_matrix:NX_FLOAT[3,3]
  polar_angle:NX_FLOAT[i]
  reflection:NX_INT[3]
  scattering_vector:NX_FLOAT
  segment_columns:NX_FLOAT
  segment_gap:NX_FLOAT
  segment_height:NX_FLOAT
  segment_rows:NX_FLOAT
  segment_thickness:NX_FLOAT
  segment_width:NX_FLOAT
  space_group:NX_CHAR
  temperature:NX_FLOAT
  temperature_coefficient:NX_FLOAT
  thickness:NX_FLOAT
  type:NX_CHAR
  unit_cell:NX_FLOAT[n_comp,6]
  unit_cell_a:NX_FLOAT
  unit_cell_alpha:NX_FLOAT
  unit_cell_b:NX_FLOAT
  unit_cell_beta:NX_FLOAT
  unit_cell_c:NX_FLOAT
  unit_cell_gamma:NX_FLOAT
  unit_cell_volume:NX_FLOAT
  usage:NX_CHAR
  wavelength:NX_FLOAT[i]
  reflectivity:NXdata
  transmission:NXdata
  NXgeometry
  temperature_log:NXlog
  shape:NXshape
```

NXcrystal Members Declarations in the *NXcrystal* group.

Name and Attributes	Type	Units	Description (and Occurrences)
usage	*NX_CHAR*		How this crystal is used. Choices are in the list. Any of these value(s): • `Bragg`: reflection geometry • `Laue`: transmission geometry, requires "thickness" field to be defined

Continued on next page

Table 2.1 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		Type or material of monochromating substance. Chemical formula can be specified separately. Use the "reflection" field to indicate the (hkl) orientation. Use the "d_spacing" field to record the lattice plane spacing. This field was changed (2010-11-17) from an enumeration to a string since common usage showed a wider variety of use than a simple list. These are the items in the list at the time of the change: PG | Ge | Si | Cu | Fe3Si | CoFe | Cu2MnAl | Multilayer | Diamond Note that "PG" is Highly Oriented Pyrolytic Graphite. Also, "Cu2MnAl" had the comment "Heusler".
chemical_formula	(*NX_CHAR*)		The chemical formula specified using CIF conventions. Abbreviated version of CIF standard: • Only recognized element symbols may be used. • Each element symbol is followed by a 'count' number. A count of '1' may be omitted. • A space or parenthesis must separate each cluster of (element symbol + count). • Where a group of elements is enclosed in parentheses, the multiplier for the group must follow the closing parentheses. That is, all element and group multipliers are assumed to be printed as subscripted numbers. • Unless the elements are ordered in a manner that corresponds to their chemical structure, the order of the elements within any group or moiety depends on whether or not carbon is present. • If carbon is present, the order should be: C, then H, then the other elements in alphabetical order of their symbol. If carbon is not present, the elements are listed purely in alphabetic order of their symbol. • This is the *Hill* system used by Chemical Abstracts.
order_no	*NX_INT*		A number which describes if this is the first, second,.. n^{th} crystal in a multi crystal monochromator
cut_angle	*NX_FLOAT*	*NX_ANGLE*	Cut angle of reflecting Bragg plane and plane of crystal surface
space_group	(*NX_CHAR*)		Space group of crystal structure

Continued on next page

Table 2.1 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
unit_cell	*NX_FLOAT*	*NX_LENGTH*	Unit cell parameters (lengths and angles) Dimensions: (rank=2) • 1: `n_comp` • 2: `6`
unit_cell_a	*NX_FLOAT*	*NX_LENGTH*	Unit cell lattice parameter: length of side a
unit_cell_b	*NX_FLOAT*	*NX_LENGTH*	Unit cell lattice parameter: length of side b
unit_cell_c	*NX_FLOAT*	*NX_LENGTH*	Unit cell lattice parameter: length of side c
unit_cell_alpha	*NX_FLOAT*	*NX_ANGLE*	Unit cell lattice parameter: angle alpha
unit_cell_beta	*NX_FLOAT*	*NX_ANGLE*	Unit cell lattice parameter: angle beta
unit_cell_gamma	*NX_FLOAT*	*NX_ANGLE*	Unit cell lattice parameter: angle gamma
unit_cell_volume	*NX_FLOAT*	*NX_VOLUME*	Volume of the unit cell
orientation_matrix	*NX_FLOAT*		Orientation matrix of single crystal sample using Busing-Levy convention Dimensions: (rank=2) • 1: `3` • 2: `3`
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	Optimum diffracted wavelength Dimensions: • 1: `i`
d_spacing	*NX_FLOAT*	*NX_LENGTH*	spacing between crystal planes of the reflection
scattering_vector	*NX_FLOAT*	*NX_WAVENUMBER*	Scattering vector, Q, of nominal reflection
reflection	*NX_INT*	*NX_UNITLESS*	Miller indices (hkl) values of nominal reflection Dimensions: • 1: `3`
thickness	*NX_FLOAT*	*NX_LENGTH*	Thickness of the crystal. (Required for Laue orientations - see "usage" field)
density	*NX_NUMBER*	*NX_MASS_DENSITY*	mass density of the crystal
segment_width	*NX_FLOAT*	*NX_LENGTH*	Horizontal width of individual segment
segment_height	*NX_FLOAT*	*NX_LENGTH*	Vertical height of individual segment
segment_thickness	*NX_FLOAT*	*NX_LENGTH*	Thickness of individual segment
segment_gap	*NX_FLOAT*	*NX_LENGTH*	Typical gap between adjacent segments
segment_columns	*NX_FLOAT*	*NX_LENGTH*	number of segment columns in horizontal direction
segment_rows	*NX_FLOAT*	*NX_LENGTH*	number of segment rows in vertical direction
mosaic_horizontal	*NX_FLOAT*	*NX_ANGLE*	horizontal mosaic Full Width Half Maximum
mosaic_vertical	*NX_FLOAT*	*NX_ANGLE*	vertical mosaic Full Width Half Maximum
curvature_horizontal	*NX_FLOAT*	*NX_ANGLE*	Horizontal curvature of focusing crystal
curvature_vertical	*NX_FLOAT*	*NX_ANGLE*	Vertical curvature of focusing crystal
is_cylindrical	*NX_BOOLEAN*		Is this crystal bent cylindrically?
cylindrical_orientation_angle	*NX_NUMBER*	*NX_ANGLE*	If cylindrical: cylinder orientation angle

Continued on next page

Table 2.1 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*NX_FLOAT*	*NX_ANGLE*	Polar (scattering) angle at which crystal assembly is positioned. Note: some instrument geometries call this term 2theta. Dimensions: • 1: `i`
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	Azimuthal angle at which crystal assembly is positioned Dimensions: • 1: `i`
bragg_angle	*NX_FLOAT*	*NX_ANGLE*	Bragg angle of nominal reflection Dimensions: • 1: `i`
temperature	*NX_FLOAT*	*NX_TEMPERATURE*	average/nominal crystal temperature
temperature_coefficient	*NX_FLOAT*	*NX_ANY*	how lattice parameter changes with temperature
(geometry)	*NXgeometry*		Position of crystal
temperature_log	*NXlog*		log file of crystal temperature
reflectivity	*NXdata*		crystal reflectivity versus wavelength
transmission	*NXdata*		crystal transmission versus wavelength
shape	*NXshape*		A NXshape group describing the shape of the crystal arrangement

NXdata

version	category	extends	groups cited
1.0	base	*NXobject*	none

(**required**) `NXdata` is a template of plottable data and their dimension scales. It is mandatory that there is at least one `NXdata` group in each NXentry group. Note that the `variable` and `data` can be defined with different names. The `signal` and `axes` attribute of the `data` item define which items are plottable data and which are *dimension scales*.

- Each `NXdata` group will consist of only one data set containing plottable data and their standard deviations.
- This data set may be of arbitrary rank up to a maximum of `NX_MAXRANK=32`.
- The plottable data will be identified by the attribute: `signal=1`
- The plottable data will identify the *dimension scale* specification(s) in the `axes` attribute.

If available, the standard deviations of the data are to be stored in a data set of the same rank and dimensions, with the name `errors`.

- For each data dimension, there should be a one-dimensional array of the same length.
- These one-dimensional arrays are the *dimension scales* of the data, *i.e.* the values of the independent variables at which the data is measured, such as scattering angle or energy transfer.

There are two methods of linking each data dimension to its respective dimension scale.

The preferred (and recommended) method uses the `axes` attribute to specify the names of each *dimension scale*.

The older method uses the `axis` attribute on each *dimension scale* to identify with an integer the axis whose value is the number of the dimension.

`NXdata` is used to implement one of the basic motivations in NeXus, to provide a default plot for the data of this `NXentry`. The actual data might be stored in another group and (hard) linked to the `NXdata` group.

symbol list: These symbols will be used below to coordinate datasets with the same shape.

dataRank rank of the `data` field

n length of the `variable` field

nx length of the `x` field

ny length of the `y` field

nz length of the `z` field

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXdata.nxdl.xml

svnid: Id: NXdata.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXdata

```
NXdata (base class, version 1.0)
  data:NX_NUMBER[n]
    @signal
    @axes
    @uncertainties
    @long_name
  errors:NX_NUMBER[n]
  offset:NX_FLOAT
  scaling_factor:NX_FLOAT
  variable:NX_NUMBER[n]
    @long_name
    @distribution
    @first_good
    @last_good
    @axis
  variable_errors:NX_NUMBER[n]
  x:NX_FLOAT[nx]
  y:NX_FLOAT[ny]
  z:NX_FLOAT[nz]
```

NXdata Members Declarations in the *NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
variable	*NX_NUMBER*		Dimension scale defining an axis of the data. Client is responsible for defining the dimensions of the data. The name of this field may be changed to fit the circumstances. Standard NeXus client tools will use the attributes to determine how to use this field. Dimensions: (rank=1) • 1: `n`
@long_name	(*NX_CHAR*)		Axis label
@distribution	*NX_BOOLEAN*		`0|false`: single value, `1|true`: multiple values
@first_good	*NX_INT*		Index of first good value
@last_good	*NX_INT*		Index of last good value
@axis	*NX_POSINT*		Index (positive integer) identifying this specific set of numbers. N.B. The `axis` attribute is the old way of designating a link. Do not use the `axes` attribute with the `axis` attribute. The `axes` attribute is now preferred.
variable_errors	*NX_NUMBER*		Errors (uncertainties) associated with axis `variable` Client is responsible for defining the dimensions of the data. The name of this field may be changed to fit the circumstances but is matched with the *variable* field with `_errors` appended. Dimensions: (rank=1) • 1: `n`
data	*NX_NUMBER*		This field contains the data values to be used as the NeXus *plottable data*. Client is responsible for defining the dimensions of the data. The name of this field may be changed to fit the circumstances. Standard NeXus client tools will use the attributes to determine how to use this field. Dimensions: (rank=dataRank) • 0: `n`
@signal	*NX_POSINT*		Plottable (independent) axis, indicate index number. Only one field in a `NXdata` group may have the `signal=1` attribute. Do not use the `signal` attribute with the `axis` attribute.
@axes	(*NX_CHAR*)		Defines the names of the dimension scales (independent axes) for this data set as a colon-delimited array. NOTE: The `axes` attribute is the preferred method of designating a link. Do not use the `axes` attribute with the `axis` attribute.
@uncertainties	(*NX_CHAR*)		Specify the names of the errors (uncertainties) of the dependent axes as plottable data. NOTE: The `errors` attribute uses the same syntax as the `axes` attribute.
@long_name	(*NX_CHAR*)		data label
errors	*NX_NUMBER*		Standard deviations of data values - the data array is identified by the attribute `signal=1`. The `errors` array must have the same dimensions as `data`. Client is responsible for defin-

NXdetector

version	category	extends	groups cited
1.1	base	*NXobject*	*NXcharacterization*, *NXdata*, *NXgeometry*, *NXnote*

Template of a detector, detector bank, or multidetector.

symbol list: These symbols will be used below to coordinate datasets with the same shape.

np number of scan points (only present in scanning measurements)

i number of detector pixels in the first (X, slowest) direction

j number of detector pixels in the second (Y, faster) direction

k number of detector pixels in the third (Z, if necessary, fastest) direction

tof number of bins in the time-of-flight histogram

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXdetector.nxdl.xml

svnid: Id: NXdetector.nxdl.xml 1231 2013-09-13 15:28:06Z Tobias Richter

Structure of NXdetector

```
NXdetector (base class, version 1.1)
  acquisition_mode:NX_CHAR
  angular_calibration:NX_FLOAT[i,j]
  angular_calibration_applied:NX_BOOLEAN
  azimuthal_angle:NX_FLOAT[np,i,j]
  beam_center_x:NX_FLOAT
  beam_center_y:NX_FLOAT
  bit_depth_readout:NX_INT
  calibration_date:NX_DATE_TIME
  count_time:NX_NUMBER[np]
  countrate_correction__applied:NX_BOOLEAN
  crate:NX_INT[i,j]
    @local_name
  data:NX_NUMBER[np,i,j,tof]
    @signal
    @axes
    @long_name
    @check_sum
    @link
  data_error:NX_NUMBER[np,i,j,tof]
    @units
    @link
  dead_time:NX_FLOAT[np,i,j]
  description:NX_CHAR
  detection_gas_path:NX_FLOAT
  detector_number:NX_INT[i,j]
  detector_readout_time:NX_FLOAT
  diameter:NX_FLOAT
  distance:NX_FLOAT[np,i,j]
  flatfield:NX_FLOAT[i,j]
  flatfield_applied:NX_BOOLEAN
  flatfield_error:NX_FLOAT[i,j]
  frame_start_number:NX_INT
  frame_time:NX_FLOAT[NP]
  gain_setting:NX_CHAR
  gas_pressure:NX_FLOAT[i,j]
  input:NX_INT[i,j]
```

```
    @local_name
  layout:NX_CHAR
  local_name:NX_CHAR
  number_of_cycles:NX_INT
  pixel_mask:NX_INT[i,j]
  pixel_mask_applied:NX_BOOLEAN
  polar_angle:NX_FLOAT[np,i,j]
  raw_time_of_flight:NX_INT[tof+1]
    @frequency
  saturation_value:NX_INT
  sensor_material:NX_CHAR
  sensor_thickness:NX_FLOAT
  sequence_number:NX_CHAR
  slot:NX_INT[i,j]
    @local_name
  solid_angle:NX_FLOAT[i,j]
  threshold_energy:NX_FLOAT
  time_of_flight:NX_FLOAT[tof+1]
    @axis
    @primary
    @long_name
    @link
  trigger_dead_time:NX_FLOAT
  trigger_delay_time:NX_FLOAT
  type:NX_CHAR
  x_pixel_offset:NX_FLOAT[i,j]
    @axis
    @primary
    @long_name
    @link
  x_pixel_size:NX_FLOAT[i,j]
  y_pixel_offset:NX_FLOAT[i,j]
    @axis
    @primary
    @long_name
  y_pixel_size:NX_FLOAT[i,j]
  NXcharacterization
  efficiency:NXdata
    efficiency:NX_FLOAT[i,j,k]
    real_time:NX_NUMBER[i,j,k]
    start_time:NX_FLOAT[np]
      @start
    stop_time:NX_FLOAT[np]
      @start
    wavelength:NX_FLOAT[i,j,k]
  NXgeometry
  calibration_method:NXnote
  data_file:NXnote
```

NXdetector Members Declarations in the *NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Total time of flight Dimensions: (rank=1) • 1: `tof+1`

Continued on next page

Table 2.2 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
@axis	*NX_POSINT*		This value: `3`
@primary	*NX_POSINT*		This value: `1`
@long_name	(*NX_CHAR*)		Axis label
@link	(*NX_CHAR*)		absolute path to location in NXdetector
raw_time_of_flight	*NX_INT*	*NX_PULSES*	In DAQ clock pulses Dimensions: (rank=1) • 1: `tof+1`
@frequency	*NX_NUMBER*		Clock frequency in Hz
detector_number	*NX_INT*		Identifier for detector Dimensions: (rank=2) • 1: `i` • 2: `j`
data	*NX_NUMBER*	*NX_ANY*	Data values from the detector. Dimensions: (rank=4) • 1: `np` • 2: `i` • 3: `j` • 4: `tof`
@signal	*NX_POSINT*		This value: `1`
@axes	(*NX_CHAR*)		`[number of scan points,x_offset?,y_offset?,time_of_f`
@long_name	(*NX_CHAR*)		Title of measurement
@check_sum	*NX_INT*		Integral of data as check of data integrity
@link	(*NX_CHAR*)		absolute path to location in NXdetector
data_error	*NX_NUMBER*	*NX_ANY*	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. Dimensions: (rank=4) • 1: `np` • 2: `i` • 3: `j` • 4: `tof`
@units	(*NX_CHAR*)		
@link	(*NX_CHAR*)		absolute path to location in NXdetector
x_pixel_offset	*NX_FLOAT*	*NX_LENGTH*	Offset from the detector center in x-direction. Can be multidimensional when needed. Dimensions: (rank=2) • 1: `i` • 2: `j`
@axis	*NX_POSINT*		This value: `1`
@primary	*NX_POSINT*		This value: `1`
@long_name	(*NX_CHAR*)		Axis label
@link	(*NX_CHAR*)		absolute path to location in NXdetector

Continued on next page

Table 2.2 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
y_pixel_offset	*NX_FLOAT*	*NX_LENGTH*	**Offset from the detector center in the y-direction.** Can be multidimensional when different values are required for each pixel. Dimensions: (rank=2) • 1: `i` • 1: `j`
@axis	*NX_POSINT*		This value: `2`
@primary	*NX_POSINT*		This value: `1`
@long_name	(*NX_CHAR*)		Axis label
distance	*NX_FLOAT*	*NX_LENGTH*	TODO: need documentation Dimensions: (rank=3) • 1: `np` • 2: `i` • 3: `j`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	TODO: need documentation Dimensions: (rank=3) • 1: `np` • 2: `i` • 3: `j`
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	TODO: need documentation Dimensions: (rank=3) • 1: `np` • 2: `i` • 3: `j`
description	(*NX_CHAR*)		name/manufacturer/model/etc. information
local_name	(*NX_CHAR*)		Local name for the detector
solid_angle	*NX_FLOAT*	*NX_SOLID_ANGLE*	Solid angle subtended by the detector at the sample Dimensions: (rank=2) • 1: `i` • 2: `j`
x_pixel_size	*NX_FLOAT*	*NX_LENGTH*	Size of each detector pixel. If it is scalar all pixels are the same size. Dimensions: (rank=2) • 1: `i` • 2: `j`
y_pixel_size	*NX_FLOAT*	*NX_LENGTH*	Size of each detector pixel. If it is scalar all pixels are the same size Dimensions: (rank=2) • 1: `i` • 2: `j`

Continued on next page

Table 2.2 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
dead_time	*NX_FLOAT*	*NX_TIME*	Detector dead time Dimensions: (rank=3) • 1: `np` • 2: `i` • 3: `j`
gas_pressure	*NX_FLOAT*	*NX_PRESSURE*	Detector gas pressure Dimensions: (rank=2) • 1: `i` • 2: `j`
detection_gas_path	*NX_FLOAT*	*NX_LENGTH*	maximum drift space dimension
crate	*NX_INT*		Crate number of detector Dimensions: (rank=2) • 1: `i` • 2: `j`
@local_name	(*NX_CHAR*)		Equivalent local term
slot	*NX_INT*		Slot number of detector Dimensions: (rank=2) • 1: `i` • 2: `j`
@local_name	(*NX_CHAR*)		Equivalent local term
input	*NX_INT*		Input number of detector Dimensions: (rank=2) • 1: `i` • 2: `j`
@local_name	(*NX_CHAR*)		Equivalent local term
type	(*NX_CHAR*)		Description of type such as He3 gas cylinder, He3 PSD, scintillator, fission chamber, proportion counter, ion chamber, ccd, pixel, image plate, CMOS, ...
calibration_date	*NX_DATE_TIME*		date of last calibration (geometry and/or efficiency) measurements
layout	(*NX_CHAR*)		How the detector is represented Any of these value(s): • `point`: • `linear`: • `area`:
count_time	*NX_NUMBER*	*NX_TIME*	Elapsed actual counting time Dimensions: (rank=1) • 1: `np`

Continued on next page

Table 2.2 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
sequence_number	(*NX_CHAR*)		In order to properly sort the order of the images taken in (for example) a tomography experiment, a sequence number is stored with each image. Dimensions: (rank=1) • 1: `nBrightFrames`
beam_center_x	*NX_FLOAT*	*NX_LENGTH*	This is the x position where the direct beam would hit the detector. This is a length, not a pixel position, and can be outside of the actual detector.
beam_center_y	*NX_FLOAT*	*NX_LENGTH*	This is the y position where the direct beam would hit the detector. This is a length, not a pixel position, and can be outside of the actual detector.
frame_start_number	*NX_INT*		This is the start number of the first frame of a scan. In PX one often scans a couple of frames on a give sample, then does something else, then returns to the same sample and scans some more frames. Each time with a new data file. This number helps concatenating such measurements.
diameter	*NX_FLOAT*	*NX_LENGTH*	The diameter of a cylindrical detector
acquisition_mode	*NX_CHAR*		The acquisition mode of the detector. Any of these value(s): • `gated`: • `triggered`: • `summed`: • `event`: • `histogrammed`:
angular_calibration_applied	*NX_BOOLEAN*		True when the angular calibration has been applied in the electronics, false otherwise.
angular_calibration	*NX_FLOAT*		Angular calibration data. Dimensions: (rank=2) • 1: `i` • 2: `j`
flatfield_applied	*NX_BOOLEAN*		True when the flat field correction has been applied in the electronics, false otherwise.
flatfield	*NX_FLOAT*		Flat field correction data. Dimensions: (rank=2) • 1: `i` • 2: `j`
flatfield_error	*NX_FLOAT*		Errors of the flat field correction data. Dimensions: (rank=2) • 1: `i` • 2: `j`

Continued on next page

Table 2.2 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
pixel_mask_applied	*NX_BOOLEAN*		True when the pixel mask correction has been applied in the electronics, false otherwise.
pixel_mask	*NX_INT*		The 32-bit pixel mask for the detector. Contains a bit field for each pixel to signal dead, blind or high or otherwise unwanted or undesirable pixels. They have the following meaning: • bit 0: gap (pixel with no sensor) • bit 1: dead • bit 2: under responding • bit 3: over responding • bit 4: noisy • bit 5: -undefined- • bit 6: pixel is part of a cluster of problematic pixels (bit set in addition to others) • bit 7: -undefined- • bit 8: user defined mask (e.g. around beamstop) • bits 9-30: -undefined- • bit 31: virtual pixel (corner pixel with interpolated value) The normal data analysis software would not take pixels into account when a bit in (mask & 0x00FF) is set. Tag bit in the upper two bytes would indicate special pixel properties that normally would not be a sole reason to reject the intensity value (unless lower bits are set as well of course). Dimensions: (rank=2) • 1: `i` • 2: `j`
countrate_correction_applied	*NX_BOOLEAN*		True when a count-rate correction has already been applied in the electronics, false otherwise.
bit_depth_readout	*NX_INT*		How many bits the electronics reads per pixel. With CCD's and single photon counting detectors, this must not align with traditional integer sizes. This can be 4, 8, 12, 14, 16, ...
detector_readout_time	*NX_FLOAT*	*NX_TIME*	Time it takes to read the detector (typically milliseconds). This is important to know for time resolved experiments.
trigger_delay_time	*NX_FLOAT*	*NX_TIME*	**Time it takes to start exposure after a trigger signa** This is important to know for time resolved experiments.

Continued on next page

Table 2.2 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
trigger_dead_time	*NX_FLOAT*	*NX_TIME*	**Time during which no new trigger signal can be ac** trigger_delay_time + exposure_time + readout_time. This is important to know for time resolved experiments.
frame_time	*NX_FLOAT*	*NX_TIME*	This is time for each frame. This is exposure_time + readout time. Dimensions: (rank=1) • 1: `NP`
gain_setting	*NX_CHAR*		The gain setting of the detector. This influences background etc. Any of these value(s): • `high`: • `standard`: • `fast`: • `auto`:
saturation_value	*NX_INT*		The value at which the detector goes into saturation. Especially common to CCD detectors, the data is known to be invalid above this value.
number_of_cycles	*NX_INT*		CCD images are sometimes constructed by summing together multiple short exposures in the electronics. This reduces background etc. This is the number of short exposures used to sum images for an image.
sensor_material	*NX_CHAR*		At times, radiation is not directly sensed by the detector. Rather, the detector might sense the output from some converter like a scintillator. This is the name of this converter material.
sensor_thickness	*NX_FLOAT*	*NX_LENGTH*	At times, radiation is not directly sensed by the detector. Rather, the detector might sense the output from some converter like a scintillator. This is the thickness of this converter material.
threshold_energy	*NX_FLOAT*	*NX_ENERGY*	Single photon counter detectors can be adjusted for a certain energy range in which they work optimally. This is the energy setting for this.
(geometry)	*NXgeometry*		Position and orientation of detector
efficiency	*NXdata*		Spectral efficiency of detector with respect to e.g. wavelength See table *NXdetector/NXdata Members*.
calibration_method	*NXnote*		summary of conversion of array data to pixels (e.g. polynomial approximations) and location of details of the calibrations
data_file	*NXnote*		
(characterization)	*NXcharacterization*		

NXdetector/NXdata Members Declarations in the *NXdetector/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
efficiency	*NX_FLOAT*	*NX_DIMENSIONLESS*	efficiency of the detector Dimensions: (rank=3) • 1: `i` • 2: `j` • 3: `k`
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	TODO: need documentation Dimensions: (rank=3) • 1: `i` • 2: `j` • 3: `k`
start_time	*NX_FLOAT*	*NX_TIME*	start time for each frame, with the `start` attribute as absolute reference Dimensions: (rank=1) • 1: `np`
@start	*NX_DATE_TIME*		
stop_time	*NX_FLOAT*	*NX_TIME*	stop time for each frame, with the `start` attribute as absolute reference Dimensions: (rank=1) • 1: `np`
@start	*NX_DATE_TIME*		
real_time	*NX_NUMBER*	*NX_TIME*	real-time of the exposure (use this if exposure time varies for each array element, otherwise use `count_time` field) Dimensions: (rank=3) • 1: `i` • 2: `j` • 3: `k`

NXdetector_group

version	category	extends	groups cited
1.0	base	*NXobject*	none

This class is used to allow a logical grouping of detector elements (e.g. which tube, bank or group of banks) to be recorded in the file. As well as allowing you to e.g just select the "left" or "east" detectors, it may also be useful for determining which elements belong to the same PSD tube and hence have e.g. the same dead time.

For example, if we had "bank1" composed of "tube1", "tube2" and "tube3" then group_names would be the string "bank1, bank1/tube1, bank1/tube2,bank1/tube3" group_index would be {1,2,3,4} group_parent would be {-1,1,1,1}

The mapping array is interpreted as group 1 is a top level group containing groups 2, 3 and 4

A `group_index` array in `NXdetector` gives the base group for a detector element.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXdetector_group.nxdl.xml

svnid: Id: NXdetector_group.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXdetector_group

```
NXdetector_group (base class, version 1.0)
  group_index:NX_INT[i]
  group_names:NX_CHAR
  group_parent:NX_INT[]
  group_type:NX_INT[]
```

NXdetector_group Members Declarations in the *NXdetector_group* group.

Name and Attributes	Type	Units	Description (and Occurrences)
group_names	(*NX_CHAR*)		Comma separated list of name
group_index	*NX_INT*		Unique ID for group. A group_index array in `NXdetector` gives the base group for a detector element. Dimensions: • 1: `i`
group_parent	*NX_INT*		Index of group parent in the hierarchy: -1 means no parent (i.e. a top level) group Dimensions: • 1: `None`
group_type	*NX_INT*		Code number for group type, e.g. bank=1, tube=2 etc. Dimensions: • 1: `None`

NXdisk_chopper

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*

TODO: need documentation

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXdisk_chopper.nxdl.xml

svnid: Id: NXdisk_chopper.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXdisk_chopper

```
NXdisk_chopper (base class, version 1.0)
  distance:NX_FLOAT
  pair_separation:NX_FLOAT
  phase:NX_FLOAT
  radius:NX_FLOAT
  ratio:NX_INT
  rotation_speed:NX_FLOAT
  slit_angle:NX_FLOAT
  slit_height:NX_FLOAT
  slits:NX_INT
  type:NX_CHAR
```

```
  wavelength_range:NX_FLOAT[2]
  NXgeometry
```

NXdisk_chopper Members Declarations in the *NXdisk_chopper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		Type of the disk-chopper: only one from the enumerated list (match text exactly) Any of these value(s): • `Chopper type single`: • `contra_rotating_pair`: • `synchro_pair`:
rotation_speed	*NX_FLOAT*	*NX_FREQUENCY*	chopper rotation speed
slits	*NX_INT*		Number of slits
slit_angle	*NX_FLOAT*	*NX_ANGLE*	angular opening
pair_separation	*NX_FLOAT*	*NX_LENGTH*	disc spacing in direction of beam
radius	*NX_FLOAT*	*NX_LENGTH*	radius to centre of slit
slit_height	*NX_FLOAT*	*NX_LENGTH*	total slit height
phase	*NX_FLOAT*	*NX_ANGLE*	chopper phase angle
ratio	*NX_INT*		pulse reduction factor of this chopper in relation to other choppers/fastest pulse in the instrument
distance	*NX_FLOAT*	*NX_LENGTH*	Effective distance to the origin
wavelength_range	*NX_FLOAT*	*NX_WAVELENGTH*	low and high values of wavelength range transmitted Dimensions: • 1: `2`
(geometry)	*NXgeometry*		

NXentry

version	category	extends	groups cited
1.0	base	*NXobject*	*NXcharacterization*, *NXcollection*, *NXdata*, *NXinstrument*, *NXmonitor*, *NXnote*, *NXparameters*, *NXprocess*, *NXsample*, *NXsubentry*, *NXuser*

(**required**) Template of the top-level NeXus group which contains all the data and associated information that comprise a single measurement. It is mandatory that there is at least one group of this type in the NeXus file.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXentry.nxdl.xml

svnid: Id: NXentry.nxdl.xml 1214 2013-05-08 12:05:24Z Pete Jemian

Structure of NXentry

```
NXentry (base class, version 1.0)
  @IDF_Version
  collection_description:NX_CHAR
  collection_identifier:NX_CHAR
  collection_time:NX_FLOAT
  definition:NX_CHAR
```

```
    @version
    @URL
  definition_local:NX_CHAR
    @version
    @URL
  duration:NX_INT
  end_time:NX_DATE_TIME
  entry_identifier:NX_CHAR
  experiment_description:NX_CHAR
  experiment_identifier:NX_CHAR
  pre_sample_flightpath:NX_FLOAT
  program_name:NX_CHAR
    @version
    @configuration
  revision:NX_CHAR
    @comment
  run_cycle:NX_CHAR
  start_time:NX_DATE_TIME
  title:NX_CHAR
  NXcharacterization
  NXcollection
  NXdata
  NXinstrument
  NXmonitor
  experiment_documentation:NXnote
  notes:NXnote
  thumbnail:NXnote
    @mime_type
  NXparameters
  NXprocess
  NXsample
  NXsubentry
  NXuser
```

NXentry Members Declarations in the *NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		Extended title for entry
experiment_identifier	(*NX_CHAR*)		Unique identifier for the experiment, defined by the facility, possibly linked to the proposals
experiment_description	(*NX_CHAR*)		Brief summary of the experiment, including key objectives.
collection_identifier	(*NX_CHAR*)		User or Data Acquisition defined group of NeXus files or NXentry
collection_description	(*NX_CHAR*)		Brief summary of the collection, including grouping criteria.
entry_identifier	(*NX_CHAR*)		unique identifier for the measurement, defined by the facility.

Continued on next page

Table 2.3 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		(alternate use: see same field in *NXsubentry* for preferred) Official NeXus NXDL schema to which this file conforms. This field is provided so that **NXentry** can be the overlay position in a NeXus data file for an application definition and its set of groups, fields, and attributes. *It is advised* to use *NXsubentry*, instead, as the overlay position.
@version	(*NX_CHAR*)		NXDL version number
@URL	(*NX_CHAR*)		URL of NXDL file
definition_local	(*NX_CHAR*)		(deprecated use: see same field in NXsubentry for preferred) Local NXDL schema extended from the file specified in the `definition` field. This contains any locally-defined, additional fields in the file.
@version	(*NX_CHAR*)		NXDL version number
@URL	(*NX_CHAR*)		URL of NXDL file
start_time	*NX_DATE_TIME*		Starting time of measurement
end_time	*NX_DATE_TIME*		Ending time of measurement
duration	*NX_INT*	*NX_TIME*	Duration of measurement
collection_time	*NX_FLOAT*	*NX_TIME*	Time transpired actually collecting data i.e. taking out time when collection was suspended due to e.g. temperature out of range
run_cycle	(*NX_CHAR*)		Such as "2007-3". Some user facilities organize their beam time into run cycles.
program_name	(*NX_CHAR*)		Name of program used to generate this file
@version	(*NX_CHAR*)		Program version number
@configuration	(*NX_CHAR*)		configuration of the program
revision	(*NX_CHAR*)		Revision id of the file due to re-calibration, reprocessing, new analysis, new instrument definition format, ...
@comment	(*NX_CHAR*)		
pre_sample_flightpath	*NX_FLOAT*	*NX_LENGTH*	This is the flightpath before the sample position. This can be determined by a chopper, by the moderator or the source itself. In other words: it the distance to the component which gives the T0 signal to the detector electronics. If another component in the NXinstrument hierarchy provides this information, this should be a link.
experiment_documentation	*NXnote*		Description of the full experiment (document in pdf, latex, ...)
notes	*NXnote*		Notes describing entry
thumbnail	*NXnote*		A small image that is representative of the entry. An example of this is a 640x480 jpeg image automatically produced by a low resolution plot of the NXdata.

Continued on next page

Table 2.3 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
@mime_type	(*NX_CHAR*)		The value should be an `image/*` This value: `image/*`
(characterization)	*NXcharacterization*		
(user)	*NXuser*		
(sample)	*NXsample*		
(instrument)	*NXinstrument*		
(collection)	*NXcollection*		
(monitor)	*NXmonitor*		
(data)	*NXdata*		
(parameters)	*NXparameters*		
(process)	*NXprocess*		
(subentry)	*NXsubentry*		

NXenvironment

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*, *NXnote*, *NXsensor*

This class describes an external condition applied to the sample

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXenvironment.nxdl.xml

svnid: Id: NXenvironment.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXenvironment

```
NXenvironment (base class, version 1.0)
  description:NX_CHAR
  name:NX_CHAR
  program:NX_CHAR
  short_name:NX_CHAR
  type:NX_CHAR
  position:NXgeometry
  NXnote
  NXsensor
```

NXenvironment Members Declarations in the *NXenvironment* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Apparatus identification code/model number; e.g. OC100 011
short_name	(*NX_CHAR*)		Alternative short name, perhaps for dashboard display like a present Seblock name
type	(*NX_CHAR*)		Type of apparatus. This could be the SE codes in scheduling database; e.g. OC/100
description	(*NX_CHAR*)		Description of the apparatus; e.g. 100mm bore orange cryostat with Roots pump
program	(*NX_CHAR*)		Program controlling the apparatus; e.g. LabView VI name
position	*NXgeometry*		The position and orientation of the apparatus
(note)	*NXnote*		Additional information, LabView logs, digital photographs, etc
(sensor)	*NXsensor*		

NXevent_data

version	category	extends	groups cited
1.0	base	*NXobject*	none

Time-of-flight events

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXevent_data.nxdl.xml

svnid: Id: NXevent_data.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXevent_data

```
NXevent_data (base class, version 1.0)
  events_per_pulse:NX_INT[j]
  pixel_number:NX_INT[i]
  pulse_height:NX_FLOAT[i,k]
  pulse_time:NX_INT[j]
    @offset
  time_of_flight:NX_INT[i]
```

NXevent_data Members Declarations in the *NXevent_data* group.

Name and Attributes	Type	Units	Description (and Occurrences)
time_of_flight	*NX_INT*	*NX_TIME_OF_FLIGHT*	A list of time of flight for each event as it comes in. This list is for all pulses with information to attach to a particular pulse located in events_per_pulse. Dimensions: (rank=1) • 1: `i`
pixel_number	*NX_INT*	*NX_DIMENSIONLESS*	There will be extra information in the NXdetector to convert pixel_number to detector_number. This list is for all pulses with information to attach to a particular pulse located in events_per_pulse. Dimensions: (rank=1) • 1: `i`
pulse_time	*NX_INT*	*NX_TIME*	The time that each pulse started with respect to the offset Dimensions: (rank=1) • 1: `j`
@offset	*NX_DATE_TIME*		ISO8601
events_per_pulse	*NX_INT*	*NX_DIMENSIONLESS*	This connects the index "i" to the index "j". The jth element is the number of events in "i" that occurred during the jth pulse. Dimensions: (rank=1) • 1: `j`
pulse_height	*NX_FLOAT*	*NX_DIMENSIONLESS*	If voltages from the ends of the detector are read out this is where they go. This list is for all events with information to attach to a particular pulse height. The information to attach to a particular pulse is located in events_per_pulse. Dimensions: (rank=2) • 1: `i` • 2: `k`

NXfermi_chopper

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*

Description of a Fermi chopper, possibly with curved slits.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXfermi_chopper.nxdl.xml

svnid: Id: NXfermi_chopper.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXfermi_chopper

```
NXfermi_chopper (base class, version 1.0)
  absorbing_material:NX_CHAR
  distance:NX_FLOAT
```

```
  energy:NX_FLOAT
  height:NX_FLOAT
  number:NX_INT
  r_slit:NX_FLOAT
  radius:NX_FLOAT
  rotation_speed:NX_FLOAT
  slit:NX_FLOAT
  transmitting_material:NX_CHAR
  type:NX_CHAR
  wavelength:NX_FLOAT
  width:NX_FLOAT
  NXgeometry
```

NXfermi_chopper Members Declarations in the *NXfermi_chopper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		Fermi chopper type
rotation_speed	*NX_FLOAT*	*NX_FREQUENCY*	chopper rotation speed
radius	*NX_FLOAT*	*NX_LENGTH*	radius of chopper
slit	*NX_FLOAT*	*NX_LENGTH*	width of an individual slit
r_slit	*NX_FLOAT*	*NX_LENGTH*	radius of curvature of slits
number	*NX_INT*	*NX_UNITLESS*	number of slits
height	*NX_FLOAT*	*NX_LENGTH*	input beam height
width	*NX_FLOAT*	*NX_LENGTH*	input beam width
distance	*NX_FLOAT*	*NX_LENGTH*	distance
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	Wavelength transmitted by chopper
energy	*NX_FLOAT*	*NX_ENERGY*	energy selected
absorbing_material	(*NX_CHAR*)		absorbing material
transmitting_material	(*NX_CHAR*)		transmitting material
(geometry)	*NXgeometry*		geometry of the fermi chopper

NXfilter

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*, *NXgeometry*, *NXlog*, *NXsensor*

Template for specifying the state of band pass filters. If uncertain whether to use `NXfilter` (band-pass filter) or `NXattenuator` (reduces beam intensity), then use `NXattenuator`.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXfilter.nxdl.xml

svnid: Id: NXfilter.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXfilter

```
NXfilter (base class, version 1.0)
  chemical_formula:NX_CHAR
  coating_material:NX_CHAR
  coating_roughness:NX_FLOAT[nsurf]
  density:NX_NUMBER
  description:NX_CHAR
  m_value:NX_FLOAT
  orientation_matrix:NX_FLOAT[n_comp,3,3]
  status:NX_CHAR
```

```
  substrate_material:NX_CHAR
  substrate_roughness:NX_FLOAT
  substrate_thickness:NX_FLOAT
  temperature:NX_FLOAT
  thickness:NX_FLOAT
  unit_cell_a:NX_FLOAT
  unit_cell_alpha:NX_FLOAT
  unit_cell_b:NX_FLOAT
  unit_cell_beta:NX_FLOAT
  unit_cell_c:NX_FLOAT
  unit_cell_gamma:NX_FLOAT
  unit_cell_volume:NX_FLOAT[n_comp]
  transmission:NXdata
  NXgeometry
  temperature_log:NXlog
  sensor_type:NXsensor
```

NXfilter Members Declarations in the *NXfilter* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		**Composition of the filter. Chemical formula can be** This field was changed (2010-11-17) from an enumeration to a string since common usage showed a wider variety of use than a simple list. These are the items in the list at the time of the change: • Beryllium • Pyrolytic Graphite • Graphite • Sapphire • Silicon • Supermirror
status	(*NX_CHAR*)		position with respect to in or out of the beam (choice of only "in" or "out") Any of these value(s): • `in`: in the beam • `out`: out of the beam
temperature	*NX_FLOAT*	*NX_TEMPERATURE*	average/nominal filter temperature
thickness	*NX_FLOAT*	*NX_LENGTH*	Thickness of the filter
density	*NX_NUMBER*	*NX_MASS_DENSITY*	mass density of the filter
chemical_formula	(*NX_CHAR*)		The chemical formula specified using CIF conventions. Abbreviated version of CIF standard: • Only recognized element symbols may be used. • Each element symbol is followed by a 'count' number. A count of '1' may be omitted. • A space or parenthesis must separate each cluster of (element symbol + count). • Where a group of elements is enclosed in parentheses, the multiplier for the group must follow the closing parentheses. That is, all element and group multipliers are assumed to be printed as subscripted numbers. • Unless the elements are ordered in a manner that corresponds to their chemical structure, the order of the elements within any group or moiety depends on whether or not carbon is present. • If carbon is present, the order should be: – C, then H, then the other elements in alphabetical order of their symbol. – If carbon is not present, the elements are listed purely in alphabetic order of their symbol. • This is the *Hill* system used by Chemical Abstracts.
unit_cell_a	*NX_FLOAT*	*NX_LENGTH*	Unit cell lattice parameter: length of side a
unit_cell_b	*NX_FLOAT*	*NX_LENGTH*	Unit cell lattice parameter: length of side b
unit_cell_c	*NX_FLOAT*	*NX_LENGTH*	Unit cell lattice parameter: length of side c
unit_cell_alpha	*NX_FLOAT*	*NX_ANGLE*	Unit cell lattice parameter: angle alpha
unit_cell_beta	*NX_FLOAT*	*NX_ANGLE*	Unit cell lattice parameter: angle beta
unit_cell_gamma	*NX_FLOAT*	*NX_ANGLE*	Unit cell lattice parameter: angle gamma

NXflipper

version	category	extends	groups cited
1.0	base	*NXobject*	none

Template of a beamline spin flipper.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXflipper.nxdl.xml

svnid: Id: NXflipper.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXflipper

```
NXflipper (base class, version 1.0)
  comp_current:NX_FLOAT
  comp_turns:NX_FLOAT
  flip_current:NX_FLOAT
  flip_turns:NX_FLOAT
  guide_current:NX_FLOAT
  guide_turns:NX_FLOAT
  thickness:NX_FLOAT
  type:NX_CHAR
```

NXflipper Members Declarations in the *NXflipper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		Any of these value(s): • `coil`: • `current-sheet`:
flip_turns	*NX_FLOAT*	*NX_PER_LENGTH*	Linear density of turns (such as number of turns/cm) in flipping field coils
comp_turns	*NX_FLOAT*	*NX_PER_LENGTH*	Linear density of turns (such as number of turns/cm) in compensating field coils
guide_turns	*NX_FLOAT*	*NX_PER_LENGTH*	Linear density of turns (such as number of turns/cm) in guide field coils
flip_current	*NX_FLOAT*	*NX_CURRENT*	Flipping field coil current in “on” state”
comp_current	*NX_FLOAT*	*NX_CURRENT*	Compensating field coil current in “on” state”
guide_current	*NX_FLOAT*	*NX_CURRENT*	Guide field coil current in “on” state”
thickness	*NX_FLOAT*	*NX_LENGTH*	thickness along path of neutron travel

NXgeometry

version	category	extends	groups cited
1.0	base	*NXobject*	*NXorientation*, *NXshape*, *NXtranslation*

This is the description for a general position of a component. It is recommended to name an instance of NXgeometry as “geometry” to aid in the use of the definition in simulation codes such as McStas. Also, in HDF, linked items must share the same name. However, it might not be possible or practical in all situations.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXgeometry.nxdl.xml

svnid: Id: NXgeometry.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXgeometry

```
NXgeometry (base class, version 1.0)
  component_index:NX_INT
  description:NX_CHAR
  NXorientation
  NXshape
  NXtranslation
```

NXgeometry Members Declarations in the *NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		Optional description/label. Probably only present if we are an additional reference point for components rather than the location of a real component
component_index	*NX_INT*		Position of the component along the beam path. The sample is at 0, components upstream have negative component_index, components downstream have positive component_index.
(shape)	*NXshape*		shape/size information of component
(translation)	*NXtranslation*		translation of component
(orientation)	*NXorientation*		orientation of component

NXguide

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*, *NXgeometry*

`NXguide` is used by neutron instruments to describe a guide consists of several mirrors building a shape through which neutrons can be guided or directed. The simplest such form is box shaped although elliptical guides are gaining in popularity. The individual parts of a guide usually have common characteristics but there are cases where they are different. For example, a neutron guide might consist of 2 or 4 coated walls or a supermirror bender with multiple, coated vanes.

To describe polarizing supermirrors such as used in neutron reflection, it may be necessary to revise this definition of `NXguide` to include `NXpolarizer` and/or `NXmirror`.

When even greater complexity exists in the definition of what constitutes a *guide*, it has been suggested that `NXguide` be redefined as a `NXcollection` of `NXmirrors` each having their own `NXgeometries` describing their location(s).

For the more general case when describing mirrors, consider using `NXmirror`.

NOTE: The NeXus International Advisory Committee welcomes comments for revision and improvement of this definition of `NXguide`.

symbol list:

nsurf number of reflecting surfaces

nwl number of wavelengths

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXguide.nxdl.xml

svnid: Id: NXguide.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXguide

```
NXguide (base class, version 1.0)
  bend_angle_x:NX_FLOAT
  bend_angle_y:NX_FLOAT
  coating_material:NX_FLOAT[nsurf]
  coating_roughness:NX_FLOAT[nsurf]
  description:NX_CHAR
  external_material:NX_CHAR
  incident_angle:NX_FLOAT
  interior_atmosphere:NX_CHAR
  m_value:NX_FLOAT[nsurf]
  number_sections:NX_INT
  substrate_material:NX_FLOAT[nsurf]
  substrate_roughness:NX_FLOAT[nsurf]
  substrate_thickness:NX_FLOAT[nsurf]
  reflectivity:NXdata
    data:NX_NUMBER[nsurf,nwl]
      @signal
      @axes
    surface:NX_NUMBER[nsurf]
    wavelength:NX_NUMBER[nwl]
  NXgeometry
```

NXguide Members Declarations in the *NXguide* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		A description of this particular instance of `NXguide`.
incident_angle	*NX_FLOAT*	*NX_ANGLE*	TODO: documentation needed
bend_angle_x	*NX_FLOAT*	*NX_ANGLE*	TODO: documentation needed
bend_angle_y	*NX_FLOAT*	*NX_ANGLE*	TODO: documentation needed
interior_atmosphere	(*NX_CHAR*)		<!– TODO –> Any of these value(s): • `vacuum`: • `helium`: • `argon`:
external_material	(*NX_CHAR*)		external material outside substrate
m_value	*NX_FLOAT*		The `m` value for a supermirror, which defines the supermirror regime in multiples of the critical angle of Nickel. Dimensions: (rank=1) • 1: `nsurf`
substrate_material	*NX_FLOAT*		TODO: documentation needed Dimensions: (rank=1) • 1: `nsurf`
substrate_thickness	*NX_FLOAT*	*NX_LENGTH*	TODO: documentation needed Dimensions: (rank=1) • 1: `nsurf`
coating_material	*NX_FLOAT*		TODO: documentation needed Dimensions: (rank=1) • 1: `nsurf`
substrate_roughness	*NX_FLOAT*	*NX_LENGTH*	TODO: documentation needed Dimensions: (rank=1) • 1: `nsurf`
coating_roughness	*NX_FLOAT*	*NX_LENGTH*	TODO: documentation needed Dimensions: (rank=1) • 1: `nsurf`
number_sections	*NX_INT*	*NX_UNITLESS*	number of substrate sections (also called `nsurf` as an index in the `NXguide` specification)
(geometry)	*NXgeometry*		TODO: Explain what this NXgeometry group means. What is intended here?
reflectivity	*NXdata*		Reflectivity as function of reflecting surface and wavelength See table *NXguide/NXdata Members*.

NXguide/NXdata Members Declarations in the *NXguide/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_NUMBER*		reflectivity of each surface as a function of wavelength Dimensions: (rank=2) • 1: `nsurf` • 2: `nwl`
@signal	*NX_POSINT*		Use `signal=1` to indicate that this is the plottable data for NeXus. This value: `1`
@axes	(*NX_CHAR*)		Use `axes="surface:wavelength"` to indicate the dimension scales to be used when plotting this data. This value: `surface:wavelength`
surface	*NX_NUMBER*	*NX_ANY*	List of surfaces. Probably best to use index numbers but the specification is very loose. Dimensions: (rank=1) • 1: `nsurf`
wavelength	*NX_NUMBER*	*NX_WAVELENGTH*	wavelengths at which reflectivity was measured Dimensions: (rank=1) • 1: `nwl`

NXinsertion_device

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*, *NXgeometry*

Description of an insertion device, as in a synchrotron.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXinsertion_device.nxdl.xml

svnid: Id: NXinsertion_device.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXinsertion_device

```
NXinsertion_device (base class, version 1.0)
  bandwidth:NX_FLOAT
  energy:NX_FLOAT
  gap:NX_FLOAT
  harmonic:NX_INT
  k:NX_FLOAT
  length:NX_FLOAT
  magnetic_wavelength:NX_FLOAT
  phase:NX_FLOAT
  poles:NX_INT
  power:NX_FLOAT
  taper:NX_FLOAT
  type:NX_CHAR
  spectrum:NXdata
  NXgeometry
```

NXinsertion_device Members Declarations in the *NXinsertion_device* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		Any of these value(s): • `undulator`: • `wiggler`:
gap	*NX_FLOAT*	*NX_LENGTH*	separation between opposing pairs of magnetic poles
taper	*NX_FLOAT*	*NX_ANGLE*	angular of gap difference between upstream and downstream ends of the insertion device
phase	*NX_FLOAT*	*NX_ANGLE*	
poles	*NX_INT*	*NX_UNITLESS*	number of poles
magnetic_wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	
k	*NX_FLOAT*	*NX_DIMENSIONLESS*	beam displacement parameter
length	*NX_FLOAT*	*NX_LENGTH*	length of insertion device
power	*NX_FLOAT*	*NX_POWER*	total power delivered by insertion device
energy	*NX_FLOAT*	*NX_ENERGY*	energy of peak intensity in output spectrum
bandwidth	*NX_FLOAT*	*NX_ENERGY*	bandwidth of peak energy
harmonic	*NX_INT*	*NX_UNITLESS*	harmonic number of peak
spectrum	*NXdata*		spectrum of insertion device
(geometry)	*NXgeometry*		"Engineering" position of insertion device

	version	category	extends	groups cited
NXinstrument	1.0	base	*NXobject*	*NXaperture*, *NXattenuator*, *NXbeam_stop*, *NXbeam*, *NXbending_magnet*, *NXcapillary*, *NXcollection*, *NXcollimator*, *NXcrystal*, *NXdetector_group*, *NXdetector*, *NXdisk_chopper*, *NXevent_data*, *NXfermi_chopper*, *NXfilter*, *NXflipper*, *NXguide*, *NXinsertion_device*, *NXmirror*, *NXmoderator*, *NXmonochromator*, *NXpolarizer*, *NXpositioner*, *NXsource*, *NXvelocity_selector*, *NXxraylens*

Template of instrument descriptions comprising various beamline components. Each component will also be a NeXus group 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.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXinstrument.nxdl.xml

svnid: Id: NXinstrument.nxdl.xml 1210 2013-05-07 16:33:56Z Pete Jemian

Structure of NXinstrument

```
NXinstrument (base class, version 1.0)
  name:NX_CHAR
    @short_name
  NXaperture
  NXattenuator
  NXbeam
  NXbeam_stop
  NXbending_magnet
  NXcapillary
  NXcollection
  NXcollimator
```

```
NXcrystal
NXdetector
NXdetector_group
NXdisk_chopper
NXevent_data
NXfermi_chopper
NXfilter
NXflipper
NXguide
NXinsertion_device
NXmirror
NXmoderator
NXmonochromator
NXpolarizer
NXpositioner
NXsource
NXvelocity_selector
NXxraylens
```

NXinstrument Members Declarations in the *NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Name of instrument
@short_name	(*NX_CHAR*)		short name for instrument, perhaps the acronym
(aperture)	*NXaperture*		
(attenuator)	*NXattenuator*		
(beam)	*NXbeam*		
(beam_stop)	*NXbeam_stop*		
(bending_magnet)	*NXbending_magnet*		
(collimator)	*NXcollimator*		
(collection)	*NXcollection*		
(capillary)	*NXcapillary*		
(crystal)	*NXcrystal*		
(detector)	*NXdetector*		
(detector_group)	*NXdetector_group*		
(disk_chopper)	*NXdisk_chopper*		
(event_data)	*NXevent_data*		
(fermi_chopper)	*NXfermi_chopper*		
(filter)	*NXfilter*		
(flipper)	*NXflipper*		
(guide)	*NXguide*		
(insertion_device)	*NXinsertion_device*		
(mirror)	*NXmirror*		
(moderator)	*NXmoderator*		
(monochromator)	*NXmonochromator*		
(polarizer)	*NXpolarizer*		
(positioner)	*NXpositioner*		
(source)	*NXsource*		
(velocity_selector)	*NXvelocity_selector*		
(xraylens)	*NXxraylens*		

NXlog

version	category	extends	groups cited
1.0	base	*NXobject*	none

Definition of information that is recorded against time, such as information monitored during the run. It contains the logged values and the times at which they were measured as elapsed time since a starting time recorded in ISO8601 format. This method of storing logged data helps to distinguish instances in which a variable is a dimension scale of the data, in which case it is stored in an `NXdata` group, and instances in which it is logged during the run, when it should be stored in an `NXlog` group. Note: When using multiple `NXlog` groups, it is suggested to place them inside a `NXcollection` group. In such cases, when `NXlog` is used in another class, `NXcollection/NXlog` is then constructed.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXlog.nxdl.xml

svnid: Id: NXlog.nxdl.xml 1144 2012-09-21 11:13:24Z Pete Jemian

Structure of NXlog

```
NXlog (base class, version 1.0)
  average_value:NX_FLOAT
  average_value_error:NX_FLOAT
  description:NX_CHAR
  duration:NX_FLOAT
  maximum_value:NX_FLOAT
  minimum_value:NX_FLOAT
  raw_value:NX_NUMBER
  time:NX_FLOAT
    @start
  value:NX_NUMBER
```

NXlog Members Declarations in the *NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
time	*NX_FLOAT*	*NX_TIME*	Time of logged entry. The times are relative to the "start" attribute and in the units specified in the "units" attribute.
@start	*NX_DATE_TIME*		
value	*NX_NUMBER*	*NX_ANY*	Array of logged value, such as temperature
raw_value	*NX_NUMBER*	*NX_ANY*	Array of raw information, such as thermocouple voltage
description	(*NX_CHAR*)		Description of logged value
average_value	*NX_FLOAT*	*NX_ANY*	
average_value_error	*NX_FLOAT*	*NX_ANY*	estimated uncertainty (often used: standard deviation) of average_value
minimum_value	*NX_FLOAT*	*NX_ANY*	
maximum_value	*NX_FLOAT*	*NX_ANY*	
duration	*NX_FLOAT*	*NX_ANY*	Total time log was taken

NXmirror

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*, *NXgeometry*, *NXshape*

Template of a beamline mirror or supermirror.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXmirror.nxdl.xml

svnid: Id: NXmirror.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXmirror

```
NXmirror (base class, version 1.0)
  bend_angle_x:NX_FLOAT
  bend_angle_y:NX_FLOAT
  coating_material:NX_CHAR
  coating_roughness:NX_FLOAT
  description:NX_CHAR
  even_layer_density:NX_FLOAT
  even_layer_material:NX_CHAR
  external_material:NX_CHAR
  incident_angle:NX_FLOAT
  interior_atmosphere:NX_CHAR
  layer_thickness:NX_FLOAT
  m_value:NX_FLOAT
  odd_layer_density:NX_FLOAT
  odd_layer_material:NX_CHAR
  substrate_density:NX_FLOAT
  substrate_material:NX_CHAR
  substrate_roughness:NX_FLOAT
  substrate_thickness:NX_FLOAT
  type:NX_CHAR
  reflectivity:NXdata
  figure_data:NXdata
  NXgeometry
  shape:NXshape
```

NXmirror Members Declarations in the *NXmirror* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		Any of these value(s): • `single`: mirror with a single material as a reflecting surface • `multi`: mirror with stacked, multiple layers as a reflecting surface
description	(*NX_CHAR*)		description of this mirror
incident_angle	*NX_FLOAT*	*NX_ANGLE*	
bend_angle_x	*NX_FLOAT*	*NX_ANGLE*	
bend_angle_y	*NX_FLOAT*	*NX_ANGLE*	
interior_atmosphere	(*NX_CHAR*)		Any of these value(s): • `vacuum`: • `helium`: • `argon`:
external_material	(*NX_CHAR*)		external material outside substrate
m_value	*NX_FLOAT*	*NX_UNITLESS*	The m value for a supermirror, which defines the supermirror regime in multiples of the critical angle of Nickel.
substrate_material	(*NX_CHAR*)		
substrate_density	*NX_FLOAT*	*NX_MASS_DENSITY*	
substrate_thickness	*NX_FLOAT*	*NX_LENGTH*	
coating_material	(*NX_CHAR*)		
substrate_roughness	*NX_FLOAT*	*NX_LENGTH*	
coating_roughness	*NX_FLOAT*	*NX_LENGTH*	
even_layer_material	(*NX_CHAR*)		
even_layer_density	*NX_FLOAT*	*NX_MASS_DENSITY*	
odd_layer_material	(*NX_CHAR*)		
odd_layer_density	*NX_FLOAT*	*NX_MASS_DENSITY*	
layer_thickness	*NX_FLOAT*	*NX_LENGTH*	An array describing the thickness of each layer
(geometry)	*NXgeometry*		<!– TODO explain what this group means –>
reflectivity	*NXdata*		Reflectivity as function of wavelength
shape	*NXshape*		A NXshape group describing the shape of the mirror
figure_data	*NXdata*		Numerical description of the surface figure of the mirror.

NXmoderator

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*, *NXgeometry*, *NXlog*

This is the description for a general moderator

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXmoderator.nxdl.xml

svnid: Id: NXmoderator.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXmoderator

```
NXmoderator (base class, version 1.0)
  coupled:NX_BOOLEAN
```

```
  coupling_material:NX_CHAR
  distance:NX_FLOAT
  poison_depth:NX_FLOAT
  poison_material:NX_CHAR
  temperature:NX_FLOAT
  type:NX_CHAR
  pulse_shape:NXdata
  NXgeometry
  temperature_log:NXlog
```

NXmoderator Members Declarations in the *NXmoderator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Effective distance as seen by measuring radiation
type	(*NX_CHAR*)		Any of these value(s): • `H20`: • `D20`: • `Liquid H2`: • `Liquid CH4`: • `Liquid D2`: • `Solid D2`: • `C`: • `Solid CH4`: • `Solid H2`:
poison_depth	*NX_FLOAT*	*NX_LENGTH*	
coupled	*NX_BOOLEAN*		whether the moderator is coupled
coupling_material	(*NX_CHAR*)		The material used for coupling. Usually Cd.
poison_material	(*NX_CHAR*)		Any of these value(s): • `Gd`: • `Cd`:
temperature	*NX_FLOAT*	*NX_TEMPERATURE*	average/nominal moderator temperature
(geometry)	*NXgeometry*		"Engineering" position of moderator
temperature_log	*NXlog*		log file of moderator temperature
pulse_shape	*NXdata*		moderator pulse shape

NXmonitor

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*, *NXlog*

Template of monitor data. It is similar to the NXdata groups containing monitor data and its associated dimension scale, e.g. time_of_flight or wavelength in pulsed neutron instruments. However, it may also include integrals, or scalar monitor counts, which are often used in both in both pulsed and steady-state instrumentation.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXmonitor.nxdl.xml

svnid: Id: NXmonitor.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXmonitor

```
NXmonitor (base class, version 1.0)
  count_time:NX_FLOAT
  data:NX_NUMBER[n]
    @signal
    @axes
  distance:NX_FLOAT
  efficiency:NX_NUMBER[]
  end_time:NX_DATE_TIME
  integral:NX_NUMBER
  mode:NX_CHAR
  preset:NX_NUMBER
  range:NX_FLOAT[2]
  sampled_fraction:NX_FLOAT
  start_time:NX_DATE_TIME
  time_of_flight:NX_FLOAT[]
  type:NX_CHAR
  NXgeometry
  integral_log:NXlog
```

NXmonitor Members Declarations in the *NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
start_time	*NX_DATE_TIME*		Starting time of measurement
end_time	*NX_DATE_TIME*		Ending time of measurement
preset	*NX_NUMBER*	*NX_ANY*	preset value for time or monitor
distance	*NX_FLOAT*	*NX_LENGTH*	Distance of monitor from sample
range	*NX_FLOAT*	*NX_ANY*	Range (X-axis, Time-of-flight, etc.) over which the integral was calculated Dimensions: • 1: `2`
integral	*NX_NUMBER*	*NX_ANY*	Total integral monitor counts
type	(*NX_CHAR*)		Any of these value(s): • `Fission Chamber`: • `Scintillator`:
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Time-of-flight Dimensions: • 1: `None`
efficiency	*NX_NUMBER*	*NX_DIMENSIONLESS*	Monitor efficiency Dimensions: • 1: `None`
data	*NX_NUMBER*	*NX_ANY*	Monitor data The signal and axes attributes take the same definitions as in *NXdata*: **signal** `signal=1` means this is the plottable data **axes** `axes="names"` where names are defined as a colon-delimited string within this attribute in the C-order of the data array Dimensions: (rank=dataRank) • 0: `n`
@signal	*NX_POSINT*		as defined for NXdata
@axes	(*NX_CHAR*)		as defined for NXdata
sampled_fraction	*NX_FLOAT*	*NX_DIMENSIONLESS*	Proportion of incident beam sampled by the monitor (0<x<1)
count_time	*NX_FLOAT*	*NX_TIME*	Elapsed actual counting time, can be an array of size `np` when scanning. This is not the difference of the calendar time but the time the instrument was really counting, without pauses or times lost due beam unavailability
integral_log	*NXlog*		Time variation of monitor counts
(geometry)	*NXgeometry*		Geometry of the monitor

NXmonochromator

version	category	extends	groups cited
1.0	base	*NXobject*	*NXcrystal*, *NXdata*, *NXgeometry*, *NXvelocity_selector*

This is a base class for everything which selects a wavelength or energy, be it a monochromator crystal, a velocity selector, an undulator or whatever.

The expected units are:

- wavelength: angstrom
- energy: eV

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXmonochromator.nxdl.xml

svnid: Id: NXmonochromator.nxdl.xml 1144 2012-09-21 11:13:24Z Pete Jemian

Structure of NXmonochromator

```
NXmonochromator (base class, version 1.0)
  energy:NX_FLOAT
  energy_error:NX_FLOAT
  wavelength:NX_FLOAT
  wavelength_error:NX_FLOAT
  NXcrystal
  distribution:NXdata
  geometry:NXgeometry
  NXvelocity_selector
```

NXmonochromator Members Declarations in the *NXmonochromator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	wavelength selected
wavelength_error	*NX_FLOAT*	*NX_WAVELENGTH*	wavelength standard deviation
energy	*NX_FLOAT*	*NX_ENERGY*	energy selected
energy_error	*NX_FLOAT*	*NX_ENERGY*	energy standard deviation
distribution	*NXdata*		
geometry	*NXgeometry*		
(crystal)	*NXcrystal*		Use as many crystals as necessary to describe
(velocity_selector)	*NXvelocity_selector*		

NXnote

version	category	extends	groups cited
1.0	base	*NXobject*	none

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

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXnote.nxdl.xml

svnid: Id: NXnote.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXnote

```
NXnote (base class, version 1.0)
  author:NX_CHAR
  data:NX_BINARY
  date:NX_DATE_TIME
  description:NX_CHAR
  file_name:NX_CHAR
  type:NX_CHAR
```

NXnote Members Declarations in the *NXnote* group.

Name and Attributes	Type	Units	Description (and Occurrences)
author	(*NX_CHAR*)		Author or creator of note
date	*NX_DATE_TIME*		Date note created/added
type	(*NX_CHAR*)		Mime content type of note data field e.g. image/jpeg, text/plain, text/html
file_name	(*NX_CHAR*)		Name of original file name if note was read from an external source
description	(*NX_CHAR*)		Title of an image or other details of the note
data	*NX_BINARY*		Binary note data - if text, line terminator is [CR][LF].

NXobject

version	category	extends	groups cited
1.0	base		none

This is the base object of NeXus

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXobject.nxdl.xml

svnid: Id: NXuser.nxdl.xml 303 2009-01-30 17:55:59Z Freddie Akeroyd

Structure of NXobject

```
NXobject (base class, version 1.0)
```

NXobject Members Declarations in the *NXobject* group.

No members to be documented

NXorientation

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*

This is the description for a general orientation of a component - it is used by the NXgeometry class

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXorientation.nxdl.xml

svnid: Id: NXorientation.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXorientation

```
NXorientation (base class, version 1.0)
  value:NX_FLOAT[numobj,6]
  NXgeometry
```

NXorientation Members Declarations in the *NXorientation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	*NX_FLOAT*	*NX_UNITLESS*	The orientation information is stored as direction cosines. The direction cosines will be between the local coordinate directions and the reference directions (to origin or relative NXgeometry). Calling the local unit vectors (x',y',z') and the reference unit vectors (x,y,z) the six numbers will be [x' dot x, x' dot y, x' dot z, y' dot x, y' dot y, y' dot z] where "dot" is the scalar dot product (cosine of the angle between the unit vectors). The unit vectors in both the local and reference coordinates are right-handed and orthonormal. The pair of groups NXtranslation and NXorientation together describe the position of a component. Dimensions: • 1: `numobj` • 2: `6`
(geometry)	*NXgeometry*		Link to another object if we are using relative positioning, else absent

NXparameters

version	category	extends	groups cited
1.0	base	*NXobject*	none

Container for parameters, usually used in processing or analysis.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXparameters.nxdl.xml

svnid: Id: NXparameters.nxdl.xml 1074 2012-05-01 13:55:54Z Freddie Akeroyd

Structure of NXparameters

```
NXparameters (base class, version 1.0)
  term:NX_CHAR
    @units
```

NXparameters Members Declarations in the *NXparameters* group.

Name and Attributes	Type	Units	Description (and Occurrences)
term	*NX_CHAR*		A parameter (also known as a term) that is used in or results from processing.
@units	(*NX_CHAR*)		

NXpolarizer

version	category	extends	groups cited
1.0	base	*NXobject*	none

Template of a beamline spin polarizer.

This is a draft and is subject to revision.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXpolarizer.nxdl.xml

svnid: Id: NXpolarizer.nxdl.xml 1200 2013-03-20 12:43:41Z Pete Jemian

Structure of NXpolarizer

```
NXpolarizer (base class, version 1.0)
  composition:NX_CHAR
  efficiency:NX_FLOAT
  reflection:NX_INT[3]
  type:NX_CHAR
```

NXpolarizer Members Declarations in the *NXpolarizer* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		one of these values: “crystal”, “supermirror”, “3He”
composition	(*NX_CHAR*)		description of the composition of the polarizing material
reflection	*NX_INT*	*NX_UNITLESS*	[hkl] values of nominal reflection Dimensions: • 1: `3`
efficiency	*NX_FLOAT*	*NX_DIMENSIONLESS*	polarizing efficiency

NXpositioner

version	category	extends	groups cited
1.0	base	*NXobject*	none

This group describes a generic positioner such as a motor or piezo-electric transducer. It is used to document the current information of a piece of beam line equipment. Note: When using multiple `NXpositioner` groups, it is suggested to place them inside a `NXcollection` group. In such cases, when `NXpositioner` is used in another class, `NXcollection/NXpositioner` is then constructed.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXpositioner.nxdl.xml

svnid: Id: NXpositioner.nxdl.xml 1144 2012-09-21 11:13:24Z Pete Jemian

Structure of NXpositioner

```
NXpositioner (base class, version 1.0)
  acceleration_time:NX_NUMBER
  controller_record:NX_CHAR
  description:NX_CHAR
  name:NX_CHAR
  raw_value:NX_NUMBER[n]
  soft_limit_max:NX_NUMBER
  soft_limit_min:NX_NUMBER
  target_value:NX_NUMBER[n]
  tolerance:NX_NUMBER[n]
  value:NX_NUMBER[n]
  velocity:NX_NUMBER
```

NXpositioner Members Declarations in the *NXpositioner* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		symbolic or mnemonic name (one word)
description	(*NX_CHAR*)		description of positioner
value	*NX_NUMBER*	*NX_ANY*	best known value of positioner - need [n] as may be scanned Dimensions: (rank=1) • 1: `n`
raw_value	*NX_NUMBER*	*NX_ANY*	raw value of positioner - need [n] as may be scanned Dimensions: (rank=1) • 1: `n`
target_value	*NX_NUMBER*	*NX_ANY*	targeted (commanded) value of positioner - need [n] as may be scanned Dimensions: (rank=1) • 1: `n`
tolerance	*NX_NUMBER*	*NX_ANY*	maximum allowable difference between target_value and value Dimensions: (rank=1) • 1: `n`
soft_limit_min	*NX_NUMBER*	*NX_ANY*	minimum allowed limit to set value
soft_limit_max	*NX_NUMBER*	*NX_ANY*	maximum allowed limit to set value
velocity	*NX_NUMBER*	*NX_ANY*	velocity of the positioner (distance moved per unit time)
acceleration_time	*NX_NUMBER*	*NX_ANY*	time to ramp the velocity up to full speed
controller_record	(*NX_CHAR*)		Hardware device record, e.g. EPICS process variable, taco/tango ...

NXprocess

version	category	extends	groups cited
1.0	base	*NXobject*	*NXnote*

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

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXprocess.nxdl.xml

svnid: Id: NXprocess.nxdl.xml 1244 2013-11-12 23:23:28Z Pete Jemian

Structure of NXprocess

```
NXprocess (base class, version 1.0)
  date:NX_DATE_TIME
  program:NX_CHAR
  version:NX_CHAR
  NXnote
```

NXprocess Members Declarations in the *NXprocess* group.

Name and Attributes	Type	Units	Description (and Occurrences)
program	*NX_CHAR*		Name of the program used
version	*NX_CHAR*		Version of the program used
date	*NX_DATE_TIME*		Date and time of processing.
(note)	*NXnote*		The note will contain information about how the data was processed or anything about the data provenance. The contents of the note can be anything that the processing code can understand, or simple text. The name will be numbered to allow for ordering of steps.

NXroot

version	category	extends	groups cited
1.0	base	*NXobject*	*NXentry*

Definition of the root NeXus group.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXroot.nxdl.xml

svnid: Id: NXroot.nxdl.xml 1150 2012-09-23 04:01:04Z Pete Jemian

Structure of NXroot

```
NXroot (base class, version 1.0)
  @NX_class
  @file_time
  @file_name
  @file_update_time
  @NeXus_version
  @HDF_version
  @HDF5_Version
  @XML_version
  @creator
  NXentry
```

NXroot Members Declarations in the *NXroot* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		entries

NXsample

version	category	extends	groups cited
1.0	base	*NXobject*	*NXbeam*, *NXdata*, *NXenvironment*, *NXgeometry*, *NXlog*, *NXpositioner*

Template of the state of the sample. This could include scanned variables that are associated with one of the data dimensions, e.g. the magnetic field, or logged data, e.g. monitored temperature vs elapsed time.

symbol list: symbolic array lengths to be coordinated between various fields

n_comp number of compositions

n_Temp number of temperatures

n_eField number of values in applied electric field

n_mField number of values in applied magnetic field

n_pField number of values in applied pressure field

n_sField number of values in applied stress field

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXsample.nxdl.xml

svnid: Id: NXsample.nxdl.xml 1214 2013-05-08 12:05:24Z Pete Jemian

Structure of NXsample

```
NXsample (base class, version 1.0)
  changer_position:NX_INT
  chemical_formula:NX_CHAR
  component:NX_CHAR
  concentration:NX_FLOAT[n_comp]
  density:NX_FLOAT[n_comp]
  description:NX_CHAR
  distance:NX_FLOAT
  electric_field:NX_FLOAT[n_eField]
    @direction
  external_DAC:NX_FLOAT
  magnetic_field:NX_FLOAT[n_mField]
    @direction
  mass:NX_FLOAT[n_comp]
  name:NX_CHAR
  orientation_matrix:NX_FLOAT[n_comp,3,3]
  path_length:NX_FLOAT
  path_length_window:NX_FLOAT
  preparation_date:NX_DATE_TIME
  pressure:NX_FLOAT[n_pField]
  relative_molecular_mass:NX_FLOAT[n_comp]
  rotation_angle:NX_FLOAT
  sample_component:NX_CHAR
  sample_orientation:NX_FLOAT[3]
  scattering_length_density:NX_FLOAT[n_comp]
  short_title:NX_CHAR
  situation:NX_CHAR
  stress_field:NX_FLOAT[n_sField]
    @direction
```

```
  temperature:NX_FLOAT[n_Temp]
  thickness:NX_FLOAT
  type:NX_CHAR
  unit_cell:NX_FLOAT[n_comp,6]
  unit_cell_class:NX_CHAR
  unit_cell_group:NX_CHAR
  unit_cell_volume:NX_FLOAT[n_comp]
  volume_fraction:NX_FLOAT[n_comp]
  x_translation:NX_FLOAT
  NXbeam
  transmission:NXdata
  temperature_env:NXenvironment
  magnetic_field_env:NXenvironment
  geometry:NXgeometry
  temperature_log:NXlog
  magnetic_field_log:NXlog
  external_ADC:NXlog
  NXpositioner
```

NXsample Members Declarations in the *NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
chemical_formula	(*NX_CHAR*)		The chemical formula specified using CIF conventions. Abbreviated version of CIF standard: • Only recognized element symbols may be used. • Each element symbol is followed by a 'count' number. A count of '1' may be omitted. • A space or parenthesis must separate each cluster of (element symbol + count). • Where a group of elements is enclosed in parentheses, the multiplier for the group must follow the closing parentheses. That is, all element and group multipliers are assumed to be printed as subscripted numbers. • Unless the elements are ordered in a manner that corresponds to their chemical structure, the order of the elements within any group or moiety depends on whether or not carbon is present. • If carbon is present, the order should be: – C, then H, then the other elements in alphabetical order of their symbol. – If carbon is not present, the elements are listed purely in alphabetic order of their symbol. • This is the *Hill* system used by Chemical Abstracts.

Continued on next page

Table 2.4 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
temperature	*NX_FLOAT*	*NX_TEMPERATURE*	Sample temperature. This could be a scanned variable Dimensions: (rank=anyRank) • 1: `n_Temp`
electric_field	*NX_FLOAT*	*NX_VOLTAGE*	Applied electric field Dimensions: • 1: `n_eField`
@direction	(*NX_CHAR*)		Any of these value(s): • `x`: • `y`: • `z`:
magnetic_field	*NX_FLOAT*	*NX_ANY*	Applied magnetic field Dimensions: • 1: `n_mField`
@direction	(*NX_CHAR*)		Any of these value(s): • `x`: • `y`: • `z`:
stress_field	*NX_FLOAT*	*NX_ANY*	Applied external stress field Dimensions: • 1: `n_sField`
@direction	(*NX_CHAR*)		Any of these value(s): • `x`: • `y`: • `z`:
pressure	*NX_FLOAT*	*NX_PRESSURE*	Applied pressure Dimensions: • 1: `n_pField`
changer_position	*NX_INT*	*NX_UNITLESS*	Sample changer position
unit_cell	*NX_FLOAT*	*NX_LENGTH*	Unit cell parameters (lengths and angles) Dimensions: (rank=2) • 1: `n_comp` • 2: `6`
unit_cell_volume	*NX_FLOAT*	*NX_VOLUME*	Volume of the unit cell Dimensions: (rank=1) • 1: `n_comp`

Continued on next page

Table 2.4 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
sample_orientation	*NX_FLOAT*	*NX_ANGLE*	This will follow the Busing and Levy convention from Acta.Crysta v22, p457 (1967) Dimensions: (rank=1) • 1: `3`
orientation_matrix	*NX_FLOAT*		Orientation matrix of single crystal sample. This will follow the Busing and Levy convention from Acta.Crysta v22, p457 (1967) Dimensions: (rank=3) • 1: `n_comp` • 2: `3` • 3: `3`
mass	*NX_FLOAT*	*NX_MASS*	Mass of sample Dimensions: (rank=1) • 1: `n_comp`
density	*NX_FLOAT*	*NX_MASS_DENSITY*	Density of sample Dimensions: (rank=1) • 1: `n_comp`
relative_molecular_mass	*NX_FLOAT*	*NX_MASS*	Relative Molecular Mass of sample Dimensions: (rank=1) • 1: `n_comp`
type	(*NX_CHAR*)		Any of these value(s): • `sample`: • `sample+can`: • `can`: • `calibration sample`: • `normalisation sample`: • `simulated data`: • `none`: • `sample environment`:
situation	(*NX_CHAR*)		The atmosphere will be one of the components, which is where its details will be stored; the relevant components will be indicated by the entry in the sample_component member. Any of these value(s): • `air`: • `vacuum`: • `inert atmosphere`: • `oxidising atmosphere`: • `reducing atmosphere`: • `sealed can`: • `other`:
description	(*NX_CHAR*)		Description of the sample
preparation_date	*NX_DATE_TIME*		Date of preparation of the sample

Continued on next page

Table 2.4 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
component	(*NX_CHAR*)		Details of the component of the sample and/or can Dimensions: (rank=1) • 1: `n_comp`
sample_component	(*NX_CHAR*)		What type of component we are: "sample | can | atmosphere | kit" Any of these value(s): • `sample`: • `can`: • `atmosphere`: • `kit`: Dimensions: (rank=1) • 1: `n_comp`
concentration	*NX_FLOAT*	*NX_MASS_DENSITY*	Concentration of each component Dimensions: (rank=1) • 1: `n_comp`
volume_fraction	*NX_FLOAT*		Volume fraction of each component Dimensions: (rank=1) • 1: `n_comp`
scattering_length_density	*NX_FLOAT*	*NX_SCATTERING_LENGTH_DENSITY*	Scattering length density of each component Dimensions: (rank=1) • 1: `n_comp`
unit_cell_class	(*NX_CHAR*)		In case it is all we know and we want to record/document it Any of these value(s): • `cubic`: • `tetragonal`: • `orthorhombic`: • `monoclinic`: • `triclinic`: Dimensions: (rank=1) • 1: `n_comp`
unit_cell_group	(*NX_CHAR*)		Crystallographic point or space group Dimensions: (rank=1) • 1: `n_comp`
path_length	*NX_FLOAT*	*NX_LENGTH*	Path length through sample/can for simple case when it does not vary with scattering direction
path_length_window	*NX_FLOAT*	*NX_LENGTH*	Thickness of a beam entry/exit window on the can (mm) - assumed same for entry and exit
thickness	*NX_FLOAT*	*NX_LENGTH*	sample thickness
external_DAC	*NX_FLOAT*	*NX_ANY*	value sent to user's sample setup
short_title	(*NX_CHAR*)		20 character fixed length sample description for legends

Continued on next page

Table 2.4 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	Optional rotation angle for the case when the powder diagram has been obtained through an omega-2theta scan like from a traditional single detector powder diffractometer
x_translation	*NX_FLOAT*	*NX_LENGTH*	Translation of the sample along the X-direction of the laboratory coordinate system
distance	*NX_FLOAT*	*NX_LENGTH*	Translation of the sample along the Z-direction of the laboratory coordinate system
geometry	*NXgeometry*		The position and orientation of the center of mass of the sample
(beam)	*NXbeam*		Details of beam incident on sample - used to calculate sample/beam interaction point
transmission	*NXdata*		As a function of Wavelength
temperature_log	*NXlog*		temperature_log.value is a link to e.g. temperature_env.sensor1.value_log.value
temperature_env	*NXenvironment*		Additional sample temperature environment information
magnetic_field_log	*NXlog*		magnetic_field_log.value is a link to e.g. magnetic_field_env.sensor1.value_log.value
magnetic_field_env	*NXenvironment*		Additional sample magnetic environment information
external_ADC	*NXlog*		logged value (or logic state) read from user's setup
(positioner)	*NXpositioner*		Any positioner (motor, PZT, ...) used to locate the sample

NXsensor

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*, *NXlog*, *NXorientation*

This class describes a sensor used to monitor an external condition - the condition itself is described in NXenvironment

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXsensor.nxdl.xml

svnid: Id: NXsensor.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXsensor

```
NXsensor (base class, version 1.0)
  attached_to:NX_CHAR
  external_field_brief:NX_CHAR
  high_trip_value:NX_FLOAT
  low_trip_value:NX_FLOAT
  measurement:NX_CHAR
  model:NX_CHAR
  name:NX_CHAR
  run_control:NX_BOOLEAN
  short_name:NX_CHAR
  type:NX_CHAR
  value:NX_FLOAT[n]
  value_deriv1:NX_FLOAT[]
```

```
14   value_deriv2:NX_FLOAT[]
15   geometry:NXgeometry
16   value_log:NXlog
17   value_deriv1_log:NXlog
18   value_deriv2_log:NXlog
19   external_field_full:NXorientation
```

NXsensor Members Declarations in the *NXsensor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
model	(*NX_CHAR*)		Sensor identification code/model number
name	(*NX_CHAR*)		Name for the sensor
short_name	(*NX_CHAR*)		Short name of sensor used e.g. on monitor display program
attached_to	(*NX_CHAR*)		where sensor is attached to ("sample" | "can")
measurement	(*NX_CHAR*)		name for measured signal Any of these value(s): • `temperature`: • `pH`: • `magnetic_field`: • `electric_field`: • `conductivity`: • `resistance`: • `voltage`: • `pressure`: • `flow`: • `stress`: • `strain`: • `shear`: • `surface_pressure`:
type	(*NX_CHAR*)		The type of hardware used for the measurement. Examples (suggestions but not restrictions): **Temperature** J | K | T | E | R | S | Pt100 | Rh/Fe **pH** Hg/Hg2Cl2 | Ag/AgCl | ISFET **Ion selective electrode** specify species; e.g. Ca2+ **Magnetic field** Hall **Surface pressure** wilhelmy plate
run_control	*NX_BOOLEAN*		Is data collection controlled or synchronised to this quantity: 1=no, 0=to "value", 1=to "value_deriv1", etc.
high_trip_value	*NX_FLOAT*	*NX_ANY*	Upper control bound of sensor reading if using run_control
low_trip_value	*NX_FLOAT*	*NX_ANY*	Lower control bound of sensor reading if using run_control
value	*NX_FLOAT*	*NX_ANY*	nominal setpoint or average value - need [n] as may be a vector Dimensions: • 1: `n`
value_deriv1	*NX_FLOAT*	*NX_ANY*	Nominal/average first derivative of value e.g. strain rate - same dimensions as "value" (may be a vector) Dimensions: • 1: `None`
value_deriv2	*NX_FLOAT*	*NX_ANY*	Nominal/average second derivative of value - same dimensions as "value" (may be a vector) Dimensions: • 1: `None`
external_field_brief	(*NX_CHAR*)		Any of these value(s): • `along beam`: • `across beam`:

NXshape

version	category	extends	groups cited
1.0	base	*NXobject*	none

This is the description of the general shape and size of a component, which may be made up of `numobj` separate elements - it is used by the *NXgeometry* class

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXshape.nxdl.xml

svnid: Id: NXshape.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXshape

```
NXshape (base class, version 1.0)
  direction:NX_CHAR
  shape:NX_CHAR
  size:NX_FLOAT[numobj,nshapepar]
```

NXshape Members Declarations in the *NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
shape	(*NX_CHAR*)		general shape of a component Any of these value(s): • `nxflat`: • `nxcylinder`: • `nxbox`: • `nxsphere`: • `nxcone`: • `nxelliptical`: • `nxtoroidal`: • `nxparabolic`: • `nxpolynomial`:
size	*NX_FLOAT*	*NX_LENGTH*	physical extent of the object along its local axes (after NXorientation) with the center of mass at the local origin (after NXtranslation). The meaning and location of these axes will vary according to the value of the "shape" variable. `nshapepar` defines how many parameters: • For "nxcylinder" type the parameters are (diameter,height) and a three value orientation vector of the cylinder. • For the "nxbox" type the parameters are (length,width,height). • For the "nxsphere" type the parameters are (diameter). • For nxcone cone half aperture • For nxelliptical, semi-major axis, semi-minor-axis, angle of major axis and pole • For nxtoroidal, major radius, minor radius • For nxparabolic, parabolic parameter a • For nxpolynomial, an array of polynom coefficients, the dimension of the array encodes the degree of the polynom Dimensions: • 1: `numobj` • 2: `nshapepar`
direction	(*NX_CHAR*)		Any of these value(s): • `concave`: • `convex`:

NXsource

version	category	extends	groups cited
1.0	base	*NXobject*	*NXdata*, *NXgeometry*, *NXnote*

Template of the neutron or x-ray source, insertion devices and/or moderators.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXsource.nxdl.xml

svnid: Id: NXsource.nxdl.xml 1064 2012-04-19 15:33:13Z Tobias Richter

Structure of NXsource

```
NXsource (base class, version 1.0)
  bunch_distance:NX_FLOAT
  bunch_length:NX_FLOAT
  current:NX_FLOAT
  distance:NX_FLOAT
  emittance_x:NX_FLOAT
  emittance_y:NX_FLOAT
  energy:NX_FLOAT
  flux:NX_FLOAT
  frequency:NX_FLOAT
  last_fill:NX_NUMBER
    @time
  mode:NX_CHAR
  name:NX_CHAR
    @short_name
  number_of_bunches:NX_INT
  period:NX_FLOAT
  power:NX_FLOAT
  probe:NX_CHAR
  pulse_width:NX_FLOAT
  sigma_x:NX_FLOAT
  sigma_y:NX_FLOAT
  target_material:NX_CHAR
  top_up:NX_BOOLEAN
  type:NX_CHAR
  voltage:NX_FLOAT
  bunch_pattern:NXdata
    title:NX_CHAR
  pulse_shape:NXdata
  distribution:NXdata
  geometry:NXgeometry
  notes:NXnote
```

NXsource Members Declarations in the *NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Effective distance from sample Distance as seen by radiation from sample. This number should be negative to signify that it is upstream of the sample.
name	(*NX_CHAR*)		Name of source
@short_name	(*NX_CHAR*)		short name for source, perhaps the acronym

Continued on next page

Table 2.5 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		type of radiation source (pick one from the enumerated list and spell exactly) Any of these value(s): • `Spallation Neutron Source`: • `Pulsed Reactor Neutron Source`: • `Reactor Neutron Source`: • `Synchrotron X-ray Source`: • `Pulsed Muon Source`: • `Rotating Anode X-ray`: • `Fixed Tube X-ray`: • `UV Laser`: • `Free-Electron Laser`: • `Optical Laser`: • `Ion Source`: • `UV Plasma Source`:
probe	(*NX_CHAR*)		type of radiation probe (pick one from the enumerated list and spell exactly) Any of these value(s): • `neutron`: • `x-ray`: • `muon`: • `electron`: • `ultraviolet`: • `visible light`: • `positron`: • `proton`:
power	*NX_FLOAT*	*NX_POWER*	Source power
emittance_x	*NX_FLOAT*	*NX_EMITTANCE*	Source emittance (nm-rad) in X (horizontal) direction.
emittance_y	*NX_FLOAT*	*NX_EMITTANCE*	Source emittance (nm-rad) in Y (horizontal) direction.
sigma_x	*NX_FLOAT*	*NX_LENGTH*	particle beam size in x
sigma_y	*NX_FLOAT*	*NX_LENGTH*	particle beam size in y
flux	*NX_FLOAT*	*NX_FLUX*	Source intensity/area (example: s-1 cm-2)
energy	*NX_FLOAT*	*NX_ENERGY*	Source energy. For storage rings, this would be the particle beam energy. For X-ray tubes, this would be the excitation voltage.
current	*NX_FLOAT*	*NX_CURRENT*	Accelerator, X-ray tube, or storage ring current
voltage	*NX_FLOAT*	*NX_VOLTAGE*	Accelerator voltage
frequency	*NX_FLOAT*	*NX_FREQUENCY*	Frequency of pulsed source
period	*NX_FLOAT*	*NX_PERIOD*	Period of pulsed source

Continued on next page

Table 2.5 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
target_material	(*NX_CHAR*)		Pulsed source target material Any of these value(s): • `Ta`: • `W`: • `depleted_U`: • `enriched_U`: • `Hg`: • `Pb`: • `C`:
number_of_bunches	*NX_INT*		For storage rings, the number of bunches in use.
bunch_length	*NX_FLOAT*	*NX_TIME*	For storage rings, temporal length of the bunch
bunch_distance	*NX_FLOAT*	*NX_TIME*	For storage rings, time between bunches
pulse_width	*NX_FLOAT*	*NX_TIME*	temporal width of source pulse
mode	(*NX_CHAR*)		source operating mode Any of these value(s): • `Single Bunch`: for storage rings • `Multi Bunch`: for storage rings
top_up	*NX_BOOLEAN*		Is the synchrotron operating in top_up mode?
last_fill	*NX_NUMBER*	*NX_CURRENT*	For storage rings, the current at the end of the most recent injection.
@time	*NX_DATE_TIME*		date and time of the most recent injection.
notes	*NXnote*		any source/facility related messages/events that occurred during the experiment
bunch_pattern	*NXdata*		For storage rings, description of the bunch pattern. This is useful to describe irregular bunch patterns. See table *NXsource/NXdata Members*.
pulse_shape	*NXdata*		source pulse shape
geometry	*NXgeometry*		"Engineering" location of source
distribution	*NXdata*		The wavelength or energy distribution of the source

NXsource/NXdata Members Declarations in the *NXsource/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		name of the bunch pattern

NXsubentry

version	category	extends	groups cited
1.0	base	*NXobject*	*NXcharacterization*, *NXcollection*, *NXdata*, *NXinstrument*, *NXmonitor*, *NXnote*, *NXparameters*, *NXprocess*, *NXsample*, *NXuser*

`NXsubentry` is a base class virtually identical to `NXentry` and is used as the (overlay) location for application definitions. Use a separate `NXsubentry` for each application definition.

To use `NXsubentry` with a hypothetical application definition called `NXmyappdef`: * Create a group with attribute `NX_class="NXsubentry"` * Within that group, create a field called `definition="NXmyappdef"`. * There are two optional attributes of definition: `version` and `URL`

The intended use is to define application definitions for a multi-technique `NXentry`. Previously, an application definition replaced `NXentry` with its own definition. With the increasing popularity of instruments combining multiple techniques for data collection (such as SAXS/WAXS instruments), it was recognized the application definitions must be entered in the NeXus data file tree as children of `NXentry`.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXsubentry.nxdl.xml

svnid: Id: NXsubentry.nxdl.xml 1214 2013-05-08 12:05:24Z Pete Jemian

Structure of NXsubentry

```
NXsubentry (base class, version 1.0)
  @IDF_Version
  collection_description:NX_CHAR
  collection_identifier:NX_CHAR
  collection_time:NX_FLOAT
  definition:NX_CHAR
    @version
    @URL
  definition_local:NX_CHAR
    @version
    @URL
  duration:NX_INT
  end_time:NX_DATE_TIME
  entry_identifier:NX_CHAR
  experiment_description:NX_CHAR
  experiment_identifier:NX_CHAR
  pre_sample_flightpath:NX_FLOAT
  program_name:NX_CHAR
    @version
    @configuration
  revision:NX_CHAR
    @comment
  run_cycle:NX_CHAR
  start_time:NX_DATE_TIME
  title:NX_CHAR
  NXcharacterization
  NXcollection
  NXdata
  NXinstrument
  NXmonitor
  experiment_documentation:NXnote
  notes:NXnote
  thumbnail:NXnote
    @mime_type
  NXparameters
  NXprocess
  NXsample
  NXuser
```

NXsubentry Members Declarations in the *NXsubentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		Extended title for entry

Continued on next page

Table 2.6 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
experiment_identifier	(*NX_CHAR*)		Unique identifier for the experiment, defined by the facility, possibly linked to the proposals
experiment_description	(*NX_CHAR*)		Brief summary of the experiment, including key objectives.
collection_identifier	(*NX_CHAR*)		User or Data Acquisition defined group of NeXus files or NXentry
collection_description	(*NX_CHAR*)		Brief summary of the collection, including grouping criteria.
entry_identifier	(*NX_CHAR*)		unique identifier for the measurement, defined by the facility.
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms
@version	(*NX_CHAR*)		NXDL version number
@URL	(*NX_CHAR*)		URL of NXDL file
definition_local	(*NX_CHAR*)		Local NXDL schema extended from the file specified in the `definition` field. This contains any locally-defined, additional fields in the file.
@version	(*NX_CHAR*)		NXDL version number
@URL	(*NX_CHAR*)		URL of NXDL file
start_time	*NX_DATE_TIME*		Starting time of measurement
end_time	*NX_DATE_TIME*		Ending time of measurement
duration	*NX_INT*	*NX_TIME*	Duration of measurement
collection_time	*NX_FLOAT*	*NX_TIME*	Time transpired actually collecting data i.e. taking out time when collection was suspended due to e.g. temperature out of range
run_cycle	(*NX_CHAR*)		Such as "2007-3". Some user facilities organize their beam time into run cycles.
program_name	(*NX_CHAR*)		Name of program used to generate this file
@version	(*NX_CHAR*)		Program version number
@configuration	(*NX_CHAR*)		configuration of the program
revision	(*NX_CHAR*)		Revision id of the file due to re-calibration, reprocessing, new analysis, new instrument definition format, ...
@comment	(*NX_CHAR*)		
pre_sample_flightpath	*NX_FLOAT*	*NX_LENGTH*	This is the flightpath before the sample position. This can be determined by a chopper, by the moderator or the source itself. In other words: it the distance to the component which gives the T0 signal to the detector electronics. If another component in the NXinstrument hierarchy provides this information, this should be a link.
experiment_documentation	*NXnote*		Description of the full experiment (document in pdf, latex, …)
notes	*NXnote*		Notes describing entry
thumbnail	*NXnote*		A small image that is representative of the entry. An example of this is a 640x480 jpeg image automatically produced by a low resolution plot of the NXdata.

Continued on next page

Table 2.6 – continued from previous page

Name and Attributes	Type	Units	Description (and Occurrences)
@mime_type	(*NX_CHAR*)		The value should be an `image/*` This value: `image/*`
(characterization)	*NXcharacterization*		
(user)	*NXuser*		
(sample)	*NXsample*		
(instrument)	*NXinstrument*		
(collection)	*NXcollection*		
(monitor)	*NXmonitor*		
(data)	*NXdata*		
(parameters)	*NXparameters*		
(process)	*NXprocess*		

NXtranslation

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*

This is the description for the general spatial location of a component - it is used by the NXgeometry class

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXtranslation.nxdl.xml

svnid: Id: NXtranslation.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXtranslation

```
NXtranslation (base class, version 1.0)
  distances:NX_FLOAT[numobj,3]
  geometry:NXgeometry
```

NXtranslation Members Declarations in the *NXtranslation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distances	*NX_FLOAT*	*NX_LENGTH*	(x,y,z) This field describes the lateral movement of a component. The pair of groups NXtranslation and NXorientation together describe the position of a component. For absolute position, the origin is the scattering center (where a perfectly aligned sample would be) with the z-axis pointing downstream and the y-axis pointing gravitationally up. For a relative position the NXtranslation is taken into account before the NXorientation. The axes are right-handed and orthonormal. Dimensions: • 1: `numobj` • 2: `3`
geometry	*NXgeometry*		Link to other object if we are relative , else absent

NXuser

version	category	extends	groups cited
1.0	base	*NXobject*	none

Template of user's contact information. The format allows more than one user with the same affiliation and contact information, but a second NXuser group should be used if they have different affiliations, etc.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXuser.nxdl.xml

svnid: Id: NXuser.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXuser

```
NXuser (base class, version 1.0)
  address:NX_CHAR
  affiliation:NX_CHAR
  email:NX_CHAR
  facility_user_id:NX_CHAR
  fax_number:NX_CHAR
  name:NX_CHAR
  role:NX_CHAR
  telephone_number:NX_CHAR
```

NXuser Members Declarations in the *NXuser* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Name of user responsible for this entry
role	(*NX_CHAR*)		Role of user responsible for this entry. Suggested roles are "local_contact", "principal_investigator", and "proposer"
affiliation	(*NX_CHAR*)		Affiliation of user
address	(*NX_CHAR*)		Address of user
telephone_number	(*NX_CHAR*)		Telephone number of user
fax_number	(*NX_CHAR*)		Fax number of user
email	(*NX_CHAR*)		Email of user
facility_user_id	(*NX_CHAR*)		facility based unique identifier for this person e.g. their identification code on the facility address/contact database

NXvelocity_selector

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*

This is the description for a (typically neutron) velocity selector

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXvelocity_selector.nxdl.xml

svnid: Id: NXvelocity_selector.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXvelocity_selector

```
NXvelocity_selector (base class, version 1.0)
  height:NX_FLOAT
  length:NX_FLOAT
  num:NX_INT
  radius:NX_FLOAT
  rotation_speed:NX_FLOAT
  spwidth:NX_FLOAT
  table:NX_FLOAT
  twist:NX_FLOAT
  type:NX_CHAR
  wavelength:NX_FLOAT
  wavelength_spread:NX_FLOAT
  width:NX_FLOAT
  geometry:NXgeometry
```

NXvelocity_selector Members Declarations in the *NXvelocity_selector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		velocity selector type
rotation_speed	*NX_FLOAT*	*NX_FREQUENCY*	velocity selector rotation speed
radius	*NX_FLOAT*	*NX_LENGTH*	radius at beam centre
spwidth	*NX_FLOAT*	*NX_LENGTH*	spoke width at beam centre
length	*NX_FLOAT*	*NX_LENGTH*	rotor length
num	*NX_INT*	*NX_UNITLESS*	number of spokes/lamella
twist	*NX_FLOAT*	*NX_ANGLE*	twist angle along axis
table	*NX_FLOAT*	*NX_ANGLE*	offset vertical angle
height	*NX_FLOAT*	*NX_LENGTH*	input beam height
width	*NX_FLOAT*	*NX_LENGTH*	input beam width
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	wavelength
wavelength_spread	*NX_FLOAT*	*NX_WAVELENGTH*	deviation FWHM /Wavelength
geometry	*NXgeometry*		

NXxraylens

version	category	extends	groups cited
1.0	base	*NXobject*	*NXnote*

This is a dictionary of field names to use for describing a X-ray lens as used at synchrotron beam lines. Based on information provided by Gerd Wellenreuther.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/base_classes/NXxraylens.nxdl.xml

svnid: Id: NXxraylens.nxdl.xml 662 2010-10-21 15:47:35Z Pete Jemian

Structure of NXxraylens

```
NXxraylens (base class, version 1.0)
  aperture:NX_FLOAT
  curvature:NX_FLOAT
  cylindrical:NX_BOOLEAN
  focus_type:NX_CHAR
  gas:NX_CHAR
  gas_pressure:NX_FLOAT
  lens_geometry:NX_CHAR
```

```
  lens_length:NX_FLOAT
  lens_material:NX_CHAR
  lens_thickness:NX_FLOAT
  number_of_lenses:NX_INT
  symmetric:NX_BOOLEAN
  cylinder_orientation:NXnote
```

NXxraylens Members Declarations in the *NXxraylens* group.

Name and Attributes	Type	Units	Description (and Occurrences)
lens_geometry	*NX_CHAR*		Geometry of the lens Any of these value(s): • `paraboloid`: • `spherical`: • `elliptical`: • `hyperbolical`:
symmetric	*NX_BOOLEAN*		Is the device symmetric?
cylindrical	*NX_BOOLEAN*		Is the device cylindrical?
focus_type	*NX_CHAR*		The type of focus of the lens Any of these value(s): • `line`: • `point`:
lens_thickness	*NX_FLOAT*	*NX_LENGTH*	Thickness of the lens
lens_length	*NX_FLOAT*	*NX_LENGTH*	Length of the lens
curvature	*NX_FLOAT*	*NX_LENGTH*	Radius of the curvature as measured in the middle of the lens
aperture	*NX_FLOAT*	*NX_LENGTH*	Diameter or radius of the lens.
number_of_lenses	*NX_INT*		Number of lenses that make up the compound lens.
lens_material	*NX_CHAR*		Material used to make the lens.
gas	*NX_CHAR*		Gas used to fill the lens
gas_pressure	*NX_FLOAT*	*NX_PRESSURE*	Gas pressure in the lens
cylinder_orientation	*NXnote*		Orientation of the cylinder axis.

Application Definitions

A description of each NeXus application definition is given. NeXus application definitions define the *minimum* set of terms that *must* be used in an instance of that class. Consider the application definitions as a *contract* between a data provider (such as the beam line control system) and a data consumer (such as a data analysis program for a scientific technique) that describes the information is certain to be available in a data file.

NXarchive

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXentry*, *NXinstrument*, *NXsample*, *NXsource*, *NXuser*

This is a definition for data to be archived by ICAT (http://www.icatproject.org/).

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXarchive.nxdl.xml

svnid: Id: NXarchive.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXarchive

```
NXarchive (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    @index
    collection_description:NX_CHAR
    collection_identifier:NX_CHAR
    collection_time:NX_FLOAT
    definition:NX_CHAR
    duration:NX_FLOAT
    end_time:NX_DATE_TIME
    entry_identifier:NX_CHAR
    experiment_description:NX_CHAR
    experiment_identifer:NX_CHAR
    program:NX_CHAR
      @version
    release_date:NX_CHAR
    revision:NX_CHAR
    run_cycle:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    instrument:NXinstrument
      description:NX_CHAR
      name:NX_CHAR
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    sample:NXsample
      chemical_formula:NX_CHAR
      description:NX_CHAR
      electric_field:NX_FLOAT
      magnetic_field:NX_FLOAT
      name:NX_CHAR
      preparation_date:NX_CHAR
      pressure:NX_FLOAT
      sample_id:NX_CHAR
      situation:NX_CHAR
      stress_field:NX_FLOAT
      temperature:NX_FLOAT
      type:NX_CHAR
    user:NXuser
      facility_user_id:NX_CHAR
      name:NX_CHAR
      role:NX_CHAR
```

NXarchive Members Declarations in the *NXarchive* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXarchive/NXentry Members*.
@index	(*NX_CHAR*)		

NXarchive/NXentry Members Declarations in the *NXarchive/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
experiment_identifer	*NX_CHAR*		unique identifier for the experiment
experiment_description	*NX_CHAR*		Brief description of the experiment and its objectives
collection_identifier	*NX_CHAR*		ID of user or DAQ define group of data files
collection_description	*NX_CHAR*		Brief summary of the collection, including grouping criteria
entry_identifier	*NX_CHAR*		unique identifier for this measurement as provided by the facility
start_time	*NX_DATE_TIME*		
end_time	*NX_DATE_TIME*		
duration	*NX_FLOAT*	*NX_TIME*	TODO: needs documentation
collection_time	*NX_FLOAT*	*NX_TIME*	TODO: needs documentation
run_cycle	*NX_CHAR*		TODO: needs documentation
revision	*NX_CHAR*		revision ID of this file, may be after recalibration, reprocessing etc.
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXarchive`
program	*NX_CHAR*		The program and version used for generating this file
@version	(*NX_CHAR*)		
release_date	*NX_CHAR*	*NX_TIME*	when this file is to be released into PD
user	*NXuser*		See table *NXarchive/NXentry/NXuser Members*.
instrument	*NXinstrument*		See table *NXarchive/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXarchive/NXentry/NXsample Members*.

NXarchive/NXentry/NXuser Members Declarations in the *NXarchive/NXentry/NXuser* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		
role	*NX_CHAR*		role of the user
facility_user_id	*NX_CHAR*		ID of the user in the facility burocracy database

NXarchive/NXentry/NXinstrument Members Declarations in the *NXarchive/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		
description	*NX_CHAR*		Brief description of the instrument
(source)	*NXsource*		See table *NXarchive/NXentry/NXinstrument/NXsource Members*.

NXarchive/NXentry/NXsample Members Declarations in the *NXarchive/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
sample_id	*NX_CHAR*		Unique database id of the sample
description	*NX_CHAR*		
type	*NX_CHAR*		Any of these value(s): • `sample`: • `sample+can`: • `calibration sample`: • `normalisation sample`: • `simulated data`: • `none`: • `sample_environment`:
chemical_formula	*NX_CHAR*		Chemical formula formatted according to CIF conventions
preparation_date	*NX_CHAR*	*NX_TIME*	
situation	*NX_CHAR*		Description of the environment the sample is in: air, vacuum, oxidizing atmosphere, dehydrated, etc.
temperature	*NX_FLOAT*	*NX_TEMPERATURE*	
magnetic_field	*NX_FLOAT*	*NX_CURRENT*	
electric_field	*NX_FLOAT*	*NX_VOLTAGE*	
stress_field	*NX_FLOAT*	*NX_UNITLESS*	
pressure	*NX_FLOAT*	*NX_PRESSURE*	

NXarchive/NXentry/NXinstrument/NXsource **Members** Declarations in the *NXarchive/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	*NX_CHAR*		Any of these value(s): • `Spallation Neutron Source`: • `Pulsed Reactor Neutron Source`: • `Reactor Neutron Source`: • `Synchrotron X-Ray Source`: • `Pulsed Muon Source`: • `Rotating Anode X-Ray`: • `Fixed Tube X-Ray`:
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXdirecttof

version	category	extends	groups cited
1.0b	application	*NXtofraw*	*NXentry*, *NXfermi_chopper*, *NXinstrument*

This is a application definition for raw data from a direct geometry TOF spectrometer

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXdirecttof.nxdl.xml

svnid: Id: NXdirecttof.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXdirecttof

```
NXdirecttof (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    NXinstrument
      fermi_chopper:NXfermi_chopper
        energy:NX_FLOAT
        rotation_speed:NX_FLOAT
```

NXdirecttof Members Declarations in the *NXdirecttof* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXdirecttof/NXentry Members*.

NXdirecttof/NXentry Members Declarations in the *NXdirecttof/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXdirecttof`
(instrument)	*NXinstrument*		See table *NXdirecttof/NXentry/NXinstrument Members*.

NXdirecttof/NXentry/NXinstrument Members Declarations in the *NXdirecttof/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
fermi_chopper	*NXfermi_chopper*		See table *NXdirecttof/NXentry/NXinstrument/NXfermi_chopper Members*.

NXdirecttof/NXentry/NXinstrument/NXfermi_chopper Members Declarations in the *NXdirecttof/NXentry/NXinstrument/NXfermi_chopper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_speed	*NX_FLOAT*	*NX_FREQUENCY*	chopper rotation speed
energy	*NX_FLOAT*	*NX_ENERGY*	energy selected

NXfluo

version	category	extends	groups cited
1.0	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXmonochromator*, *NXsample*, *NXsource*

This is an application definition for raw data from an X-ray fluorescence experiment

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXfluo.nxdl.xml

svnid: Id: NXfluo.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXfluo

```
NXfluo (application definition, version 1.0)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /entry/instrument/fluorecence/data
      energy --> /entry/instrument/fluorecence/energy
    NXinstrument
      fluorescence:NXdetector
        data:NX_INT[nenergy]
        energy:NX_FLOAT[nenergy]
      monochromator:NXmonochromator
        wavelength:NX_FLOAT
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    NXmonitor
      data:NX_INT
      mode:NX_CHAR
      preset:NX_FLOAT
    NXsample
      name:NX_CHAR
```

NXfluo Members Declarations in the *NXfluo* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXfluo/NXentry Members*.

NXfluo/NXentry Members Declarations in the *NXfluo/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXfluo`
(instrument)	*NXinstrument*		See table *NXfluo/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXfluo/NXentry/NXsample Members*.
(monitor)	*NXmonitor*		See table *NXfluo/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXfluo/NXentry/NXdata Members*.

NXfluo/NXentry/NXinstrument Members Declarations in the *NXfluo/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXfluo/NXentry/NXinstrument/NXsource Members*.
monochromator	*NXmonochromator*		See table *NXfluo/NXentry/NXinstrument/NXmonochro Members*.
fluorescence	*NXdetector*		See table *NXfluo/NXentry/NXinstrument/NXdetector Members*.

NXfluo/NXentry/NXsample Members Declarations in the *NXfluo/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample

NXfluo/NXentry/NXmonitor Members Declarations in the *NXfluo/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
data	*NX_INT*		

NXfluo/NXentry/NXdata Members Declarations in the *NXfluo/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
energy	*link*		target = `/entry/instrument/fluorecence/e`
data	*link*		target = `/entry/instrument/fluorecence/d`

NXfluo/NXentry/NXinstrument/NXsource Members Declarations in the *NXfluo/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		This value: `x-ray`

NXfluo/NXentry/NXinstrument/NXmonochromator Members Declarations in the *NXfluo/NXentry/NXinstrument/NXmonochromator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
wavelength	*NX_FLOAT*		

NXfluo/NXentry/NXinstrument/NXdetector Members Declarations in the *NXfluo/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=1) • 1: `nenergy`
energy	*NX_FLOAT*		Dimensions: (rank=1) • 1: `nenergy`

NXindirecttof

version	category	extends	groups cited
1.0b	application	*NXtofraw*	*NXentry*, *NXinstrument*, *NXmonochromator*

This is a application definition for raw data from a direct geometry TOF spectrometer

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXindirecttof.nxdl.xml

svnid: Id: NXindirecttof.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXindirecttof

```
NXindirecttof (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    NXinstrument
      analyser:NXmonochromator
        distance:NX_FLOAT[ndet]
        energy:NX_FLOAT[nDet]
        polar_angle:NX_FLOAT[ndet]
```

NXindirecttof Members Declarations in the *NXindirecttof* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXindirecttof/NXentry Members*.

NXindirecttof/NXentry Members Declarations in the *NXindirecttof/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXindirecttof`
(instrument)	*NXinstrument*		See table *NXindirecttof/NXentry/NXinstrument Members*.

NXindirecttof/NXentry/NXinstrument Members Declarations in the *NXindirecttof/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
analyser	*NXmonochromator*		See table *NXindirect-tof/NXentry/NXinstrument/NXmonochromator Members*.

NXindirecttof/NXentry/NXinstrument/NXmonochromator Members Declarations in the *NXindirect-tof/NXentry/NXinstrument/NXmonochromator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
energy	*NX_FLOAT*	*NX_ENERGY*	analyzed energy Dimensions: (rank=1) • 1: `nDet`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	polar angle towards sample Dimensions: (rank=1) • 1: `ndet`
distance	*NX_FLOAT*	*NX_LENGTH*	distance from sample Dimensions: (rank=1) • 1: `ndet`

NXiqproc

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXentry*, *NXinstrument*, *NXparameters*, *NXprocess*, *NXsample*, *NXsource*

Actually this is a template from which to start an application definition.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXiqproc.nxdl.xml

svnid: Id: NXiqproc.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXiqproc

```
NXiqproc (application definition, version 1.0b)
  (overlays NXentry)
  NXentry
    @entry
    definition:NX_CHAR
    title:NX_CHAR
    NXdata
      data:NX_INT[NE,NQX,NQY]
      qx:NX_CHAR
      qy:NX_CHAR
      variable:NX_CHAR
        @varied_variable
    instrument:NXinstrument
      name:NX_CHAR
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    reduction:NXprocess
      program:NX_CHAR
```

```
      version:NX_CHAR
      input:NXparameters
        filenames:NX_CHAR
      output:NXparameters
    NXsample
      name:NX_CHAR
```

NXiqproc Members Declarations in the *NXiqproc* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXiqproc/NXentry Members*.
@entry	(*NX_CHAR*)		

NXiqproc/NXentry Members Declarations in the *NXiqproc/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXiqproc`
instrument	*NXinstrument*		See table *NXiqproc/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXiqproc/NXentry/NXsample Members*.
reduction	*NXprocess*		See table *NXiqproc/NXentry/NXprocess Members*.
(data)	*NXdata*		See table *NXiqproc/NXentry/NXdata Members*.

NXiqproc/NXentry/NXinstrument Members Declarations in the *NXiqproc/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		Name of the instrument from which this data was reduced.
(source)	*NXsource*		See table *NXiqproc/NXentry/NXinstrument/NXsource Members*.

NXiqproc/NXentry/NXsample Members Declarations in the *NXiqproc/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample

NXiqproc/NXentry/NXprocess Members Declarations in the *NXiqproc/NXentry/NXprocess* group.

Name and Attributes	Type	Units	Description (and Occurrences)
program	*NX_CHAR*		
version	*NX_CHAR*		
input	*NXparameters*		Input parameters for the reduction program used See table *NXiqproc/NXentry/NXprocess/NXparameters Members*.
output	*NXparameters*		Eventual output parameters from the data reduction program used

NXiqproc/NXentry/NXdata Members Declarations in the *NXiqproc/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		This is I(Q). The client has to analyse the dimensions of I(Q). Often, multiple I(Q) for various environment conditions are measured; that would be the first dimension. Q can be multidimensional, this accounts for the further dimensions in the data Dimensions: (rank=3) • 1: `NE` • 2: `NQX` • 3: `NQY`
variable	(*NX_CHAR*)		Dimensions: (rank=1) • 1: `NE`
@varied_variable	(*NX_CHAR*)		The real name of the varied variable in the first dim of data, temperature, P, MF etc...
qx	(*NX_CHAR*)		Values for the first dimension of Q Dimensions: (rank=1) • 1: `NQX`
qy	(*NX_CHAR*)		Values for the second dimension of Q Dimensions: (rank=1) • 1: `NQY`

NXiqproc/NXentry/NXinstrument/NXsource Members Declarations in the *NXiqproc/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXiqproc/NXentry/NXprocess/NXparameters Members Declarations in the *NXiqproc/NXentry/NXprocess/NXparameters* group.

Name and Attributes	Type	Units	Description (and Occurrences)
filenames	*NX_CHAR*		Raw data files used to generate this I(Q)

NXlauetof

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*

This is the application definition for a TOF laue diffractometer

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXlauetof.nxdl.xml

svnid: Id: NXlauetof.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXlauetof

```
NXlauetof (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    name:NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      time_of_flight --> /NXentry/NXinstrument/NXdetector/time_of_flight
    instrument:NXinstrument
      detector:NXdetector
        azimuthal_angle:NX_FLOAT
        data:NX_INT[number of x pixels,number of y pixels,nTOF]
          @signal
        distance:NX_FLOAT
        polar_angle:NX_FLOAT
        time_of_flight:NX_FLOAT[nTOF]
        x_pixel_size:NX_FLOAT
        y_pixel_size:NX_FLOAT
    control:NXmonitor
      data:NX_INT[nTOF]
      mode:NX_CHAR
      preset:NX_FLOAT
      time_of_flight:NX_FLOAT[nTOF]
    sample:NXsample
      name:NX_CHAR
      orientation_matrix:NX_FLOAT[3,3]
      unit_cell:NX_FLOAT[6]
```

NXlauetof Members Declarations in the *NXlauetof* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXlauetof/NXentry Members*.

NXlauetof/NXentry Members Declarations in the *NXlauetof/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXlauetof`
instrument	*NXinstrument*		See table *NXlauetof/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXlauetof/NXentry/NXsample Members*.
control	*NXmonitor*		See table *NXlauetof/NXentry/NXmonitor Members*.
name	*NXdata*		See table *NXlauetof/NXentry/NXdata Members*.

NXlauetof/NXentry/NXinstrument Members Declarations in the *NXlauetof/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		This assumes a planar 2D detector. All angles and distances refer to the center of the detector. See table *NXlauetof/NXentry/NXinstrument/NXdetecto Members*.

NXlauetof/NXentry/NXsample Members Declarations in the *NXlauetof/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
orientation_matrix	*NX_FLOAT*		The orientation matrix according to Busing and Levy conventions. This is not strictly necessary as the UB can always be derived from the data. But let us bow to common usage which includes thie UB nearly always. Dimensions: (rank=2) • 1: `3` • 2: `3`
unit_cell	*NX_FLOAT*		The unit cell, a, b, c, alpha, beta, gamma. Again, not strictly necessary, but normally written. Dimensions: (rank=1) • 1: `6`

NXlauetof/NXentry/NXmonitor Members Declarations in the *NXlauetof/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		**Count to a preset value based on either clock time (** (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
data	*NX_INT*		use these attributes `primary=1 signal=1` Dimensions: (rank=1) • 1: `nTOF`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `nTOF`

NXlauetof/NXentry/NXdata Members Declarations in the *NXlauetof/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXlauetof/NXentry/NXinstrument/NXdetector Members Declarations in the *NXlauetof/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*NX_FLOAT*	*NX_ANGLE*	The polar_angle (two theta) where the detector is placed.
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	The azimuthal angle where the detector is placed.
data	*NX_INT*		Dimensions: (rank=3) • 1: `number of x pixels` • 2: `number of y pixels` • 3: `nTOF`
@signal	*NX_POSINT*		This value: `1`
x_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
y_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
distance	*NX_FLOAT*	*NX_LENGTH*	
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `nTOF`

NXmonopd

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXcrystal*, *NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*, *NXsource*

Monochromatic Neutron and X-Ray Powder Diffraction. Instrument definition for a powder diffractometer at a monochromatic neutron or X-ray beam. This is both suited for a powder diffractometer with a single detector or a powder diffractometer with a position sensitive detector.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXmonopd.nxdl.xml

svnid: Id: NXmonopd.nxdl.xml 1172 2012-09-28 16:47:09Z Pete Jemian

Structure of NXmonopd

```
NXmonopd (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      polar_angle --> /NXentry/NXinstrument/NXdetector/polar_angle
    NXinstrument
      NXcrystal
        wavelength:NX_FLOAT[i]
      NXdetector
        data:NX_INT[ndet]
        polar_angle:NX_FLOAT[ndet]
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    NXmonitor
      integral:NX_FLOAT
      mode:NX_CHAR
      preset:NX_FLOAT
    NXsample
      name:NX_CHAR
      rotation_angle:NX_FLOAT
```

NXmonopd Members Declarations in the *NXmonopd* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXmonopd/NXentry Members*.

NXmonopd/NXentry Members Declarations in the *NXmonopd/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXmonopd`
(instrument)	*NXinstrument*		See table *NXmonopd/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXmonopd/NXentry/NXsample Members*.
(monitor)	*NXmonitor*		See table *NXmonopd/NXentry/NXmonitor Members*.
(data)	*NXdata*		See table *NXmonopd/NXentry/NXdata Members*.

NXmonopd/NXentry/NXinstrument Members Declarations in the *NXmonopd/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXmonopd/NXentry/NXinstrument/NXsource Members*.
(crystal)	*NXcrystal*		See table *NXmonopd/NXentry/NXinstrument/NXcrystal Members*.
(detector)	*NXdetector*		See table *NXmonopd/NXentry/NXinstrument/NXdetector Members*.

NXmonopd/NXentry/NXsample Members Declarations in the *NXmonopd/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	Optional rotation angle for the case when the powder diagram has been obtained through an omega-2theta scan like from a traditional single detector powder diffractometer

NXmonopd/NXentry/NXmonitor Members Declarations in the *NXmonopd/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
integral	*NX_FLOAT*	*NX_ANY*	Total integral monitor counts

NXmonopd/NXentry/NXdata Members Declarations in the *NXmonopd/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*link*		target = `/NXentry/NXinstrument/NXdetecto` Link to polar angle in /NXentry/NXinstrument/NXdetector
data	*link*		target = `/NXentry/NXinstrument/NXdetecto` Link to data in /NXentry/NXinstrument/NXdetector

NXmonopd/NXentry/NXinstrument/NXsource Members Declarations in the *NXmonopd/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXmonopd/NXentry/NXinstrument/NXcrystal **Members** Declarations in the *NXmonopd/NXentry/NXinstrument/NXcrystal* group.

Name and Attributes	Type	Units	Description (and Occurrences)
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	Optimum diffracted wavelength Dimensions: (rank=1) • 1: `i`

NXmonopd/NXentry/NXinstrument/NXdetector **Members** Declarations in the *NXmonopd/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*NX_FLOAT*		where ndet = number of detectors Dimensions: (rank=1) • 1: `ndet`
data	*NX_INT*		detector signal (usually counts) are already corrected for detector efficiency Dimensions: (rank=1) • 1: `ndet`

NXrefscan

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXmonochromator*, *NXsample*, *NXsource*

This is an application definition for a monochromatic scanning reflectometer. It does not have the information to calculate the resolution since it does not have any apertures.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXrefscan.nxdl.xml

svnid: Id: NXrefscan.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXrefscan

```
NXrefscan (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    end_time:NX_DATE_TIME
    start_time:NX_DATE_TIME
```

```
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      polar_angle --> /NXentry/NXinstrument/NXdetector/polar_angle
      rotation_angle --> /NXentry/NXsample/rotation_angle
    instrument:NXinstrument
      NXdetector
        data:NX_INT[NP]
        polar_angle:NX_FLOAT[NP]
      monochromator:NXmonochromator
        wavelength:NX_FLOAT
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    control:NXmonitor
      data:NX_FLOAT[NP]
      mode:NX_CHAR
      preset:NX_FLOAT
    sample:NXsample
      name:NX_CHAR
      rotation_angle:NX_FLOAT[NP]
```

NXrefscan Members Declarations in the *NXrefscan* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXrefscan/NXentry Members*.

NXrefscan/NXentry Members Declarations in the *NXrefscan/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
end_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXrefscan`
instrument	*NXinstrument*		See table *NXrefscan/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXrefscan/NXentry/NXsample Members*.
control	*NXmonitor*		See table *NXrefscan/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXrefscan/NXentry/NXdata Members*.

NXrefscan/NXentry/NXinstrument Members Declarations in the *NXrefscan/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXrefscan/NXentry/NXinstrument/NXsource Members*.
monochromator	*NXmonochromator*		See table *NXrefscan/NXentry/NXinstrument/NXmonochromator Members*.
(detector)	*NXdetector*		See table *NXrefscan/NXentry/NXinstrument/NXdetector Members*.

NXrefscan/NXentry/NXsample Members Declarations in the *NXrefscan/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `NP`

NXrefscan/NXentry/NXmonitor Members Declarations in the *NXrefscan/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
data	*NX_FLOAT*	*NX_ANY*	Monitor counts for each step Dimensions: (rank=1) • 1: `NP`

NXrefscan/NXentry/NXdata Members Declarations in the *NXrefscan/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
rotation_angle	*link*		target = `/NXentry/NXsample/rotation_angl`
polar_angle	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXrefscan/NXentry/NXinstrument/NXsource Members Declarations in the *NXrefscan/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXrefscan/NXentry/NXinstrument/NXmonochromator Members Declarations in the *NXrefscan/NXentry/NXinstrument/NXmonochromator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	

NXrefscan/NXentry/NXinstrument/NXdetector Members Declarations in the *NXrefscan/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=1) • 1: `NP`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `NP`

NXreftof

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXdisk_chopper*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsan*

This is an application definition for raw data from a TOF reflectometer.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXreftof.nxdl.xml

svnid: Id: NXreftof.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXreftof

```
NXreftof (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    end_time:NX_DATE_TIME
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      time_binning --> /NXentry/NXinstrument/NXdetector/time_binning
    instrument:NXinstrument
      name:NX_CHAR
      detector:NXdetector
        data:NX_INT[xsize,ysize,nTOF]
        distance:NX_FLOAT
        polar_angle:NX_FLOAT
        time_of_flight:NX_FLOAT[nTOF]
        x_pixel_size:NX_FLOAT
        y_pixel_size:NX_FLOAT
      chopper:NXdisk_chopper
        distance:NX_FLOAT
    control:NXmonitor
      data:NX_INT
      integral:NX_INT
      mode:NX_CHAR
      preset:NX_FLOAT
```

```
      time_of_flight:NX_FLOAT
    sample:NXsample
      name:NX_CHAR
      rotation_angle:NX_FLOAT
```

NXreftof Members Declarations in the *NXreftof* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXreftof/NXentry Members*.

NXreftof/NXentry Members Declarations in the *NXreftof/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
end_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXreftof`
instrument	*NXinstrument*		See table *NXreftof/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXreftof/NXentry/NXsample Members*.
control	*NXmonitor*		See table *NXreftof/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXreftof/NXentry/NXdata Members*.

NXreftof/NXentry/NXinstrument Members Declarations in the *NXreftof/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		
chopper	*NXdisk_chopper*		See table *NXreftof/NXentry/NXinstrument/NXdisk_cho Members*.
detector	*NXdetector*		See table *NXreftof/NXentry/NXinstrument/NXdetector Members*.

NXreftof/NXentry/NXsample Members Declarations in the *NXreftof/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	

NXreftof/NXentry/NXmonitor Members Declarations in the *NXreftof/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*	*NX_ANY*	preset value for time or monitor
integral	*NX_INT*		Total integral monitor counts
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Time channels
data	*NX_INT*		Monitor counts in each time channel

NXreftof/NXentry/NXdata Members Declarations in the *NXreftof/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
time_binning	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXreftof/NXentry/NXinstrument/NXdisk_chopper Members Declarations in the *NXreftof/NXentry/NXinstrument/NXdisk_chopper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Distance between chopper and sample

NXreftof/NXentry/NXinstrument/NXdetector Members Declarations in the *NXreftof/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=3) • 1: `xsize` • 2: `ysize` • 3: `nTOF`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Array of time values for each bin in a time-of-flight measurement Dimensions: (rank=1) • 1: `nTOF`
distance	*NX_FLOAT*	*NX_LENGTH*	
polar_angle	*NX_FLOAT*	*NX_ANGLE*	
x_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
y_pixel_size	*NX_FLOAT*	*NX_LENGTH*	

NXsas

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXcollimator*, *NXdata*, *NXdetector*, *NXentry*, *NXgeometry*, *NXinstrument*, *NXmonitor*, *NXmonochromator*, *NXsample*, *NXshape*, *NXsource*

This is an application definition for raw data (not processed or reduced data) from a 2-D small angle scattering instrument collected with a monochromatic beam and an area detector. It is meant to be suitable both for neutron SANS and X-ray SAXS data.

It covers all raw data from all SAS techniques: SAS, WSAS, grazing incidence, GISAS

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXsas.nxdl.xml

svnid: Id: NXsas.nxdl.xml 1165 2012-09-27 18:12:40Z Pete Jemian

Structure of NXsas

```
NXsas (application definition, version 1.0b)
  (overlays NXentry)
  NXentry
    @entry
    definition:NX_CHAR
    end_time:NX_DATE_TIME
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
    instrument:NXinstrument
      name:NX_CHAR
      collimator:NXcollimator
        geometry:NXgeometry
          shape:NXshape
            shape:NX_CHAR
            size:NX_FLOAT
      detector:NXdetector
        aequatorial_angle:NX_FLOAT
        azimuthal_angle:NX_FLOAT
        beam_center_x:NX_FLOAT
        beam_center_y:NX_FLOAT
        data:NX_NUMBER[nXPixel,nYPixel]
        distance:NX_FLOAT
        polar_angle:NX_FLOAT
        rotation_angle:NX_FLOAT
        x_pixel_size:NX_FLOAT
        y_pixel_size:NX_FLOAT
      monochromator:NXmonochromator
        wavelength:NX_FLOAT
        wavelength_spread:NX_FLOAT
      source:NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    control:NXmonitor
      integral:NX_FLOAT
      mode:NX_CHAR
      preset:NX_FLOAT
    sample:NXsample
      aequatorial_angle:NX_FLOAT
      name:NX_CHAR
```

NXsas Members Declarations in the *NXsas* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXsas/NXentry Members*.
@entry	(*NX_CHAR*)		NeXus convention is to use `entry1`, `entry2`, ... for analysis software to locate each entry

NXsas/NXentry Members Declarations in the *NXsas/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
end_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXsas`
instrument	*NXinstrument*		See table *NXsas/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXsas/NXentry/NXsample Members*.
control	*NXmonitor*		See table *NXsas/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXsas/NXentry/NXdata Members*.

NXsas/NXentry/NXinstrument Members Declarations in the *NXsas/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		Name of the instrument actually used to perform the experiment
source	*NXsource*		See table *NXsas/NXentry/NXinstrument/NXsource Members*.
monochromator	*NXmonochromator*		See table *NXsas/NXentry/NXinstrument/NXmonochromator Members*.
collimator	*NXcollimator*		See table *NXsas/NXentry/NXinstrument/NXcollimator Members*.
detector	*NXdetector*		See table *NXsas/NXentry/NXinstrument/NXdetector Members*.

NXsas/NXentry/NXsample Members Declarations in the *NXsas/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
aequatorial_angle	*NX_FLOAT*	*NX_ANGLE*	

NXsas/NXentry/NXmonitor Members Declarations in the *NXsas/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
integral	*NX_FLOAT*	*NX_ANY*	Total integral monitor counts

NXsas/NXentry/NXdata Members Declarations in the *NXsas/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXsas/NXentry/NXinstrument/NXsource Members Declarations in the *NXsas/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		type of radiation source
name	(*NX_CHAR*)		Name of the radiation source
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`:

NXsas/NXentry/NXinstrument/NXmonochromator Members Declarations in the *NXsas/NXentry/NXinstrument/NXmonochromator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	The wavelength of the radiation
wavelength_spread	*NX_FLOAT*		delta_lambda/lambda ($\Delta\lambda/\lambda$): Important for resolution calculations

NXsas/NXentry/NXinstrument/NXcollimator Members Declarations in the *NXsas/NXentry/NXinstrument/NXcollimator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
geometry	*NXgeometry*		See table *NXsas/NXentry/NXinstrument/NXcollimator/ Members*.

NXsas/NXentry/NXinstrument/NXdetector Members Declarations in the *NXsas/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_NUMBER*		This is area detector data, of number of x-pixel versus number of y-pixels. Since the beam center is to be determined as a step of data reduction, it is not necessary to document or assume the position of the beam center in acquired data. Dimensions: (rank=2) • 1: `nXPixel` • 2: `nYPixel`
distance	*NX_FLOAT*	*NX_LENGTH*	The distance between detector and sample
x_pixel_size	*NX_FLOAT*	*NX_LENGTH*	Physical size of a pixel in x-direction
y_pixel_size	*NX_FLOAT*	*NX_LENGTH*	Size of a pixel in y direction
polar_angle	*NX_FLOAT*	*NX_ANGLE*	
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	
aequatorial_angle	*NX_FLOAT*	*NX_ANGLE*	
beam_center_x	*NX_FLOAT*	*NX_LENGTH*	This is the x position where the direct beam would hit the detector. This is a length, not a pixel position, and can be outside of the actual detector.
beam_center_y	*NX_FLOAT*	*NX_LENGTH*	This is the y position where the direct beam would hit the detector. This is a length, not a pixel position, and can be outside of the actual detector.

NXsas/NXentry/NXinstrument/NXcollimator/NXgeometry **Members** Declarations in the *NXsas/NXentry/NXinstrument/NXcollimator/NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
shape	*NXshape*		See table *NXsas/NXentry/NXinstrument/NXcollimator/NXgeometry/NXshape Members*.

NXsas/NXentry/NXinstrument/NXcollimator/NXgeometry/NXshape **Members** Declarations in the *NXsas/NXentry/NXinstrument/NXcollimator/NXgeometry/NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
shape	*NX_CHAR*		Any of these value(s): • `nxcylinder`: • `nxbox`:
size	*NX_FLOAT*	*NX_LENGTH*	The collimation length

NXsastof

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXcollimator*, *NXdata*, *NXdetector*, *NXentry*, *NXgeometry*, *NXinstrument*, *NXmonitor*, *NXsample*, *NXshape*, *NXsource*

This is an application definition for small angle scattering using a 2D detector in TOF mode. It strives to cover all the SAS techniques in the file again

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXsastof.nxdl.xml

svnid: Id: NXsastof.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXsastof

```
NXsastof (application definition, version 1.0b)
  (overlays NXentry)
  NXentry
    @entry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      time_of_flight --> /NXentry/NXinstrument/NXdetector/time_of_flight
    instrument:NXinstrument
      name:NX_CHAR
      collimator:NXcollimator
        geometry:NXgeometry
          shape:NXshape
            shape:NX_CHAR
            size:NX_FLOAT
      detector:NXdetector
        aequatorial_angle:NX_FLOAT
        azimuthal_angle:NX_FLOAT
        beam_center_x:NX_FLOAT
        beam_center_y:NX_FLOAT
        data:NX_NUMBER[nXPixel,nYPixel,nTOF]
        distance:NX_FLOAT
        polar_angle:NX_FLOAT
        rotation_angle:NX_FLOAT
        time_of_flight:NX_FLOAT[nTOF]
        x_pixel_size:NX_FLOAT
        y_pixel_size:NX_FLOAT
      source:NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    control:NXmonitor
      data:NX_INT[nTOF]
      mode:NX_CHAR
      preset:NX_FLOAT
      time_of_flight:NX_FLOAT[nTOF]
    sample:NXsample
      aequatorial_angle:NX_FLOAT
      name:NX_CHAR
```

NXsastof Members Declarations in the *NXsastof* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXsastof/NXentry Members*.
@entry	(*NX_CHAR*)		NeXus convention is to use “entry1”, “entry2”, ... for analysis software to locate each entry

NXsastof/NXentry Members Declarations in the *NXsastof/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXsastof`
instrument	*NXinstrument*		See table *NXsastof/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXsastof/NXentry/NXsample Members*.
control	*NXmonitor*		See table *NXsastof/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXsastof/NXentry/NXdata Members*.

NXsastof/NXentry/NXinstrument Members Declarations in the *NXsastof/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		Name of the instrument actually used to perform the experiment
source	*NXsource*		See table *NXsastof/NXentry/NXinstrument/NXsource Members*.
collimator	*NXcollimator*		See table *NXsastof/NXentry/NXinstrument/NXcollimat Members*.
detector	*NXdetector*		See table *NXsastof/NXentry/NXinstrument/NXdetector Members*.

NXsastof/NXentry/NXsample Members Declarations in the *NXsastof/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
aequatorial_angle	*NX_FLOAT*	*NX_ANGLE*	

NXsastof/NXentry/NXmonitor Members Declarations in the *NXsastof/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
data	*NX_INT*		Dimensions: (rank=1) • 1: `nTOF`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `nTOF`

NXsastof/NXentry/NXdata Members Declarations in the *NXsastof/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXsastof/NXentry/NXinstrument/NXsource Members Declarations in the *NXsastof/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		type of radiation source
name	(*NX_CHAR*)		Name of the radiation source
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`:

NXsastof/NXentry/NXinstrument/NXcollimator Members Declarations in the *NXsastof/NXentry/NXinstrument/NXcollimator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
geometry	*NXgeometry*		See table *NXsastof/NXentry/NXinstrument/NXcollimat Members*.

NXsastof/NXentry/NXinstrument/NXdetector Members Declarations in the *NXsastof/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_NUMBER*		This is area detector data, of number of x-pixel versus number of y-pixels. Since the beam center is to be determined as a step of data reduction, it is not necessary to document or assume the position of the beam center in acquired data. Dimensions: (rank=3) • 1: `nXPixel` • 2: `nYPixel` • 3: `nTOF`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `nTOF`
distance	*NX_FLOAT*	*NX_LENGTH*	The distance between detector and sample
x_pixel_size	*NX_FLOAT*	*NX_LENGTH*	Physical size of a pixel in x-direction
y_pixel_size	*NX_FLOAT*	*NX_LENGTH*	Size of a pixel in y direction
polar_angle	*NX_FLOAT*	*NX_ANGLE*	
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	
aequatorial_angle	*NX_FLOAT*	*NX_ANGLE*	
beam_center_x	*NX_FLOAT*	*NX_LENGTH*	This is the x position where the direct beam would hit the detector. This is a length, not a pixel position, and can be outside of the actual detector.
beam_center_y	*NX_FLOAT*	*NX_LENGTH*	This is the y position where the direct beam would hit the detector. This is a length, not a pixel position, and can be outside of the actual detector.

NXsastof/NXentry/NXinstrument/NXcollimator/NXgeometry Members Declarations in the *NXsastof/NXentry/NXinstrument/NXcollimator/NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
shape	*NXshape*		See table *NXsastof/NXentry/NXinstrument/NXcollimat Members*.

NXsastof/NXentry/NXinstrument/NXcollimator/NXgeometry/NXshape Members Declarations in the *NXsastof/NXentry/NXinstrument/NXcollimator/NXgeometry/NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
shape	*NX_CHAR*		Any of these value(s): • `nxcylinder`: • `nxbox`:
size	*NX_FLOAT*	*NX_LENGTH*	The collimation length

NXscan

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*

Application definition for a generic scan instrument. This definition is more an example then a stringent definition as the content of a given NeXus scan file needs to differ for different types of scans. This example definition shows a scan like done on a rotation camera: the sample is rotated and a detector image, the rotation angle and a monitor value

is stored at each step in the scan. In the following, the symbol `NP` is used to represent the number of scan points. These are the rules for storing scan data in NeXus files which are implemented in this example:

- **Each value varied throughout a scan is stored as an array of** length `NP` at its respective location within the NeXus hierarchy.
- **For area detectors, `NP` is the first dimension,** example for a detector of 256x256: `data[NP,256,256]`
- **The NXdata group contains links to all variables varied in the scan and the data.** This to give an equivalent to the more familiar classical tabular representation of scans.

These rules exist for a reason: HDF allows the first dimension of a data set to be unlimited. This means the data can be appended too. Thus a NeXus file built according to the rules given above can be used in the following way:

- At the start of a scan, write all the static information.
- **At each scan point, append new data from varied variables** and the detector to the file.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXscan.nxdl.xml

svnid: Id: NXscan.nxdl.xml 1144 2012-09-21 11:13:24Z Pete Jemian

Structure of NXscan

```
NXscan (application definition, version 1.0b)
  (overlays NXentry)
  NXentry
    definition:NX_CHAR
    end_time:NX_DATE_TIME
    start_time:NX_DATE_TIME
    title:NX_CHAR
    NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      rotation_angle --> /NXentry/NXsample/rotation_angle
    NXinstrument
      NXdetector
        data:NX_INT[NP,xdim,ydim]
    NXmonitor
      data:NX_INT[NP]
    NXsample
      rotation_angle:NX_FLOAT[NP]
```

NXscan Members Declarations in the *NXscan* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXscan/NXentry Members*.

NXscan/NXentry Members Declarations in the *NXscan/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
end_time	*NX_DATE_TIME*		
definition	*NX_CHAR*		Official NeXus NXDL schema to which this file conforms This value: `NXscan`
(instrument)	*NXinstrument*		See table *NXscan/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXscan/NXentry/NXsample Members*.
(monitor)	*NXmonitor*		See table *NXscan/NXentry/NXmonitor Members*.
(data)	*NXdata*		See table *NXscan/NXentry/NXdata Members*.

NXscan/NXentry/NXinstrument Members Declarations in the *NXscan/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(detector)	*NXdetector*		See table *NXscan/NXentry/NXinstrument/NXdetector Members*.

NXscan/NXentry/NXsample Members Declarations in the *NXscan/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_angle	*NX_FLOAT*		Dimensions: (rank=1) • 1: `NP`

NXscan/NXentry/NXmonitor Members Declarations in the *NXscan/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=1) • 1: `NP`

NXscan/NXentry/NXdata Members Declarations in the *NXscan/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
rotation_angle	*link*		target = `/NXentry/NXsample/rotation_angl`

NXscan/NXentry/NXinstrument/NXdetector Members Declarations in the *NXscan/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=3) • 1: `NP` • 2: `xdim` • 3: `ydim`

NXspe

version	category	extends	groups cited
1.0	application	*NXobject*	*NXcollection*, *NXdata*, *NXentry*, *NXfermi_chopper*, *NXinstrument*, *NXsample*

NXSPE Inelastic Format. Application definition for NXSPE file format.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXspe.nxdl.xml

svnid: None

Structure of NXspe

```
NXspe (application definition, version 1.0)
  (overlays NXentry)
  NXentry
    definition:NX_CHAR
      @version
    program_name:NX_CHAR
    NXSPE_info:NXcollection
      fixed_energy:NX_FLOAT
      ki_over_kf_scaling:NX_BOOLEAN
      psi:NX_FLOAT
    data:NXdata
      azimuthal:NX_FLOAT
      azimuthal_width:NX_FLOAT
      data:NX_NUMBER
      distance:NX_FLOAT
      energy:NX_FLOAT
      error:NX_NUMBER
      polar:NX_FLOAT
      polar_width:NX_FLOAT
    NXinstrument
      name:NX_CHAR
      NXfermi_chopper
        energy:NX_NUMBER
    NXsample
      rotation_angle:NX_NUMBER
      seblock:NX_CHAR
      temperature:NX_NUMBER
```

NXspe Members Declarations in the *NXspe* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXspe/NXentry Members*.

NXspe/NXentry Members Declarations in the *NXspe/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
program_name	(*NX_CHAR*)		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms. Any of these value(s): • `NXSPE`: • `NXspe`:
@version	(*NX_CHAR*)		
NXSPE_info	*NXcollection*		See table *NXspe/NXentry/NXcollection Members*.
data	*NXdata*		See table *NXspe/NXentry/NXdata Members*.
(instrument)	*NXinstrument*		See table *NXspe/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXspe/NXentry/NXsample Members*.

NXspe/NXentry/NXcollection Members Declarations in the *NXspe/NXentry/NXcollection* group.

Name and Attributes	Type	Units	Description (and Occurrences)
fixed_energy	*NX_FLOAT*	*NX_ENERGY*	The fixed energy used for this file.
ki_over_kf_scaling	*NX_BOOLEAN*		Indicates whether ki/kf scaling has been applied or not.
psi	*NX_FLOAT*	*NX_ANGLE*	Orientation angle as expected in DCS-MSlice

NXspe/NXentry/NXdata Members Declarations in the *NXspe/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
azimuthal	*NX_FLOAT*	*NX_ANGLE*	
azimuthal_width	*NX_FLOAT*	*NX_ANGLE*	
polar	*NX_FLOAT*	*NX_ANGLE*	
polar_width	*NX_FLOAT*	*NX_ANGLE*	
distance	*NX_FLOAT*	*NX_LENGTH*	
data	*NX_NUMBER*		
error	*NX_NUMBER*		
energy	*NX_FLOAT*	*NX_ENERGY*	

NXspe/NXentry/NXinstrument Members Declarations in the *NXspe/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		
(fermi_chopper)	*NXfermi_chopper*		See table *NXspe/NXentry/NXinstrument/NXfermi_chop Members*.

NXspe/NXentry/NXsample Members Declarations in the *NXspe/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_angle	*NX_NUMBER*	*NX_ANGLE*	
seblock	*NX_CHAR*		
temperature	*NX_NUMBER*	*NX_TEMPERATURE*	

NXspe/NXentry/NXinstrument/NXfermi_chopper Members Declarations in the *NXspe/NXentry/NXinstrument/NXfermi_chopper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
energy	*NX_NUMBER*	*NX_ENERGY*	

NXsqom

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXentry*, *NXinstrument*, *NXparameters*, *NXprocess*, *NXsample*, *NXsource*

This is the application definition for S(Q,OM) processed data. As this kind of data is in general not on a rectangular grid after data reduction, it is stored as Q,E positions plus their intensity, table like. It is the task of a possible visualisation program to regrid this data in a sensible way.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXsqom.nxdl.xml

svnid: Id: NXsqom.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXsqom

```
NXsqom (application definition, version 1.0b)
  (overlays NXentry)
  NXentry
    @entry
    definition:NX_CHAR
    title:NX_CHAR
    NXdata
      data:NX_INT[NP]
      en:NX_FLOAT[NP]
      qx:NX_CHAR
      qy:NX_CHAR
      qz:NX_CHAR
    instrument:NXinstrument
      name:NX_CHAR
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    reduction:NXprocess
      program:NX_CHAR
      version:NX_CHAR
      input:NXparameters
        filenames:NX_CHAR
      output:NXparameters
    NXsample
      name:NX_CHAR
```

NXsqom Members Declarations in the *NXsqom* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXsqom/NXentry Members*.
@entry	(*NX_CHAR*)		

NXsqom/NXentry Members Declarations in the *NXsqom/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXsqom`
instrument	*NXinstrument*		See table *NXsqom/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXsqom/NXentry/NXsample Members*.
reduction	*NXprocess*		See table *NXsqom/NXentry/NXprocess Members*.
(data)	*NXdata*		See table *NXsqom/NXentry/NXdata Members*.

NXsqom/NXentry/NXinstrument Members Declarations in the *NXsqom/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		Name of the instrument from which this data was reduced.
(source)	*NXsource*		See table *NXsqom/NXentry/NXinstrument/NXsource Members*.

NXsqom/NXentry/NXsample Members Declarations in the *NXsqom/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample

NXsqom/NXentry/NXprocess Members Declarations in the *NXsqom/NXentry/NXprocess* group.

Name and Attributes	Type	Units	Description (and Occurrences)
program	*NX_CHAR*		
version	*NX_CHAR*		
input	*NXparameters*		Input parameters for the reduction program used See table *NXsqom/NXentry/NXprocess/NXparameters Members*.
output	*NXparameters*		Eventual output parameters from the data reduction program used

NXsqom/NXentry/NXdata Members Declarations in the *NXsqom/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		This is the intensity for each point in QE Dimensions: (rank=1) • 1: `NP`
qx	(*NX_CHAR*)	*NX_WAVENUMBER*	Positions for the first dimension of Q Dimensions: (rank=1) • 1: `NP`
qy	(*NX_CHAR*)	*NX_WAVENUMBER*	Positions for the the second dimension of Q Dimensions: (rank=1) • 1: `NP`
qz	(*NX_CHAR*)	*NX_WAVENUMBER*	Positions for the the third dimension of Q Dimensions: (rank=1) • 1: `NP`
en	*NX_FLOAT*	*NX_ENERGY*	Values for the energy transfer for each point Dimensions: (rank=1) • 1: `NP`

NXsqom/NXentry/NXinstrument/NXsource Members Declarations in the *NXsqom/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXsqom/NXentry/NXprocess/NXparameters Members Declarations in the *NXsqom/NXentry/NXprocess/NXparameters* group.

Name and Attributes	Type	Units	Description (and Occurrences)
filenames	*NX_CHAR*		Raw data files used to generate this I(Q)

NXtas

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXcrystal*, *NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*, *NXsource*

This is an application definition for a triple axis spectrometer. It is for the trademark scan of the TAS, the Q-E scan. For your alignment scans use the rules in NXscan.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXtas.nxdl.xml

svnid: Id: NXtas.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXtas

```
NXtas (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      ef --> /NXentry/NXinstrument/analyzer:NXcrystal/ef
      ei --> /NXentry/NXinstrument/monochromator:NXcrystal/ei
      en --> /NXentry/NXsample/en
      qh --> /NXentry/NXsample/qh
      qk --> /NXentry/NXsample/qk
      ql --> /NXentry/NXsample/ql
    NXinstrument
      monochromator:NXcrystal
        ei:NX_FLOAT[np]
        rotation_angle:NX_FLOAT[np]
      analyser:NXcrystal
        ef:NX_FLOAT[np]
        polar_angle:NX_FLOAT[np]
        rotation_angle:NX_FLOAT[np]
      NXdetector
        data:NX_INT[np]
        polar_angle:NX_FLOAT[np]
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
    NXmonitor
      data:NX_FLOAT[np]
      mode:NX_CHAR
      preset:NX_FLOAT
    NXsample
      en:NX_FLOAT[np]
      name:NX_CHAR
      orientation_matrix:NX_FLOAT[9]
      polar_angle:NX_FLOAT[np]
      qh:NX_FLOAT[np]
      qk:NX_FLOAT[np]
      ql:NX_FLOAT[np]
      rotation_angle:NX_FLOAT[np]
      sgl:NX_FLOAT[np]
      sgu:NX_FLOAT[np]
      unit_cell:NX_FLOAT[6]
```

NXtas Members Declarations in the *NXtas* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXtas/NXentry Members*.

NXtas/NXentry Members Declarations in the *NXtas/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	*NX_CHAR*		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXtas`
(instrument)	*NXinstrument*		See table *NXtas/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXtas/NXentry/NXsample Members*.
(monitor)	*NXmonitor*		See table *NXtas/NXentry/NXmonitor Members*.
(data)	*NXdata*		One of the ei,ef,qh,qk,ql,en should get a primary=1 attribute to denote the main scan axis See table *NXtas/NXentry/NXdata Members*.

NXtas/NXentry/NXinstrument Members Declarations in the *NXtas/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXtas/NXentry/NXinstrument/NXsource Members*.
monochromator	*NXcrystal*		See table *NXtas/NXentry/NXinstrument/NXcrystal Members*.
analyser	*NXcrystal*		See table *NXtas/NXentry/NXinstrument/NXcrystal Members*.
(detector)	*NXdetector*		See table *NXtas/NXentry/NXinstrument/NXdetector Members*.

NXtas/NXentry/NXsample Members Declarations in the *NXtas/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
qh	*NX_FLOAT*	*NX_DIMENSIONLESS*	Dimensions: (rank=1) • 1: `np`
qk	*NX_FLOAT*	*NX_DIMENSIONLESS*	Dimensions: (rank=1) • 1: `np`
ql	*NX_FLOAT*	*NX_DIMENSIONLESS*	Dimensions: (rank=1) • 1: `np`
en	*NX_FLOAT*	*NX_ENERGY*	Dimensions: (rank=1) • 1: `np`
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `np`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `np`
sgu	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `np`
sgl	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `np`
unit_cell	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `6`
orientation_matrix	*NX_FLOAT*	*NX_DIMENSIONLESS*	Dimensions: (rank=1) • 1: `9`

NXtas/NXentry/NXmonitor Members Declarations in the *NXtas/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
data	*NX_FLOAT*	*NX_ANY*	Total integral monitor counts Dimensions: (rank=1) • 1: `np`

NXtas/NXentry/NXdata Members Declarations in the *NXtas/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
ei	*link*		target = `/NXentry/NXinstrument/monochrom`
ef	*link*		target = `/NXentry/NXinstrument/analyzer:`
en	*link*		target = `/NXentry/NXsample/en`
qh	*link*		target = `/NXentry/NXsample/qh`
qk	*link*		target = `/NXentry/NXsample/qk`
ql	*link*		target = `/NXentry/NXsample/ql`
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXtas/NXentry/NXinstrument/NXsource Members Declarations in the *NXtas/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`:

NXtas/NXentry/NXinstrument/NXcrystal Members Declarations in the *NXtas/NXentry/NXinstrument/NXcrystal* group.

Name and Attributes	Type	Units	Description (and Occurrences)
ei	*NX_FLOAT*	*NX_ENERGY*	Dimensions: (rank=1) • 1: `np`
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `np`

NXtas/NXentry/NXinstrument/NXcrystal Members Declarations in the *NXtas/NXentry/NXinstrument/NXcrystal* group.

Name and Attributes	Type	Units	Description (and Occurrences)
ef	*NX_FLOAT*	*NX_ENERGY*	Dimensions: (rank=1) • 1: `np`
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `np`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `np`

NXtas/NXentry/NXinstrument/NXdetector Members Declarations in the *NXtas/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=1) • 1: `np`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `np`

NXtofnpd

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*, *NXuser*

This is a application definition for raw data from a TOF neutron powder diffractometer

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXtofnpd.nxdl.xml

svnid: Id: NXtofnpd.nxdl.xml 1130 2012-09-19 16:02:54Z Freddie Akeroyd

Structure of NXtofnpd

```
NXtofnpd (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    pre_sample_flightpath:NX_FLOAT
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      detector_number --> /NXentry/NXinstrument/NXdetector/detector_number
      time_of_flight --> /NXentry/NXinstrument/NXdetector/time_of_flight
    NXinstrument
      detector:NXdetector
        azimuthal_angle:NX_FLOAT[ndet]
        data:NX_INT[ndet,ntimechan]
        detector_number:NX_INT[ndet]
        distance:NX_FLOAT[ndet]
        polar_angle:NX_FLOAT[ndet]
        time_of_flight:NX_FLOAT[ntimechan]
    NXmonitor
      data:NX_INT[ntimechan]
      distance:NX_FLOAT
      mode:NX_CHAR
      preset:NX_FLOAT
      time_of_flight:NX_FLOAT[ntimechan]
    NXsample
      name:NX_CHAR
    user:NXuser
      name:NX_CHAR
```

NXtofnpd Members Declarations in the *NXtofnpd* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXtofnpd/NXentry Members*.

NXtofnpd/NXentry Members Declarations in the *NXtofnpd/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXtofnpd`
pre_sample_flightpath	*NX_FLOAT*	*NX_LENGTH*	This is the flight path before the sample position. This can be determined by a chopper, by the moderator or the source itself. In other words: it the distance to the component which gives the T0 signal to the detector electronics. If another component in the NXinstrument hierarchy provides this information, this should be a link.
user	*NXuser*		See table *NXtofnpd/NXentry/NXuser Members*.
(instrument)	*NXinstrument*		See table *NXtofnpd/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXtofnpd/NXentry/NXsample Members*.
(monitor)	*NXmonitor*		See table *NXtofnpd/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXtofnpd/NXentry/NXdata Members*.

NXtofnpd/NXentry/NXuser Members Declarations in the *NXtofnpd/NXentry/NXuser* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		

NXtofnpd/NXentry/NXinstrument Members Declarations in the *NXtofnpd/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		See table *NXtofnpd/NXentry/NXinstrument/NXdetector Members*.

NXtofnpd/NXentry/NXsample Members Declarations in the *NXtofnpd/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample

NXtofnpd/NXentry/NXmonitor Members Declarations in the *NXtofnpd/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
distance	*NX_FLOAT*	*NX_LENGTH*	
data	*NX_INT*		Dimensions: (rank=1) • 1: `ntimechan`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `ntimechan`

NXtofnpd/NXentry/NXdata Members Declarations in the *NXtofnpd/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
detector_number	*link*		target = `/NXentry/NXinstrument/NXdetecto`
time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXtofnpd/NXentry/NXinstrument/NXdetector Members Declarations in the *NXtofnpd/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=2) • 1: `ndet` • 2: `ntimechan`
detector_number	*NX_INT*		Dimensions: (rank=1) • 1: `ndet`
distance	*NX_FLOAT*	*NX_LENGTH*	distance to sample for each detector Dimensions: (rank=1) • 1: `ndet`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `ntimechan`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	polar angle for each detector element Dimensions: (rank=1) • 1: `ndet`
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	azimuthal angle for each detector element Dimensions: (rank=1) • 1: `ndet`

NXtofraw

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*, *NXuser*

This is an application definition for raw data from a generic TOF instrument

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXtofraw.nxdl.xml

svnid: Id: NXtofraw.nxdl.xml 1130 2012-09-19 16:02:54Z Freddie Akeroyd

Structure of NXtofraw

```
NXtofraw (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    duration:NX_FLOAT
    pre_sample_flightpath:NX_FLOAT
    run_number:NX_INT
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      detector_number --> /NXentry/NXinstrument/NXdetector/detector_number
      time_of_flight --> /NXentry/NXinstrument/NXdetector/time_of_flight
    instrument:NXinstrument
      detector:NXdetector
        azimuthal_angle:NX_FLOAT[ndet]
        data:NX_INT[ndet,ntimechan]
        detector_number:NX_INT[ndet]
        distance:NX_FLOAT[ndet]
        polar_angle:NX_FLOAT[ndet]
        time_of_flight:NX_FLOAT[ntimechan]
    NXmonitor
      data:NX_INT[ntimechan]
      distance:NX_FLOAT
      integral_counts:NX_INT
      mode:NX_CHAR
      preset:NX_FLOAT
      time_of_flight:NX_FLOAT[ntimechan]
    NXsample
      name:NX_CHAR
      nature:NX_CHAR
    user:NXuser
      name:NX_CHAR
```

NXtofraw Members Declarations in the *NXtofraw* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXtofraw/NXentry Members*.

NXtofraw/NXentry Members Declarations in the *NXtofraw/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXtofraw`
duration	*NX_FLOAT*		
run_number	*NX_INT*		
pre_sample_flightpath	*NX_FLOAT*	*NX_LENGTH*	This is the flight path before the sample position. This can be determined by a chopper, by the moderator, or the source itself. In other words: it is the distance to the component which gives the T0 signal to the detector electronics. If another component in the NXinstrument hierarchy provides this information, this should be a link.
user	*NXuser*		See table *NXtofraw/NXentry/NXuser Members*.
instrument	*NXinstrument*		See table *NXtofraw/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXtofraw/NXentry/NXsample Members*.
(monitor)	*NXmonitor*		See table *NXtofraw/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXtofraw/NXentry/NXdata Members*.

NXtofraw/NXentry/NXuser Members Declarations in the *NXtofraw/NXentry/NXuser* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		

NXtofraw/NXentry/NXinstrument Members Declarations in the *NXtofraw/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		See table *NXtofraw/NXentry/NXinstrument/NXdetector Members*.

NXtofraw/NXentry/NXsample Members Declarations in the *NXtofraw/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
nature	*NX_CHAR*		Any of these value(s): • `powder`: • `liquid`: • `single crystal`:

NXtofraw/NXentry/NXmonitor Members Declarations in the *NXtofraw/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
distance	*NX_FLOAT*	*NX_LENGTH*	
data	*NX_INT*		Dimensions: (rank=1) • 1: `ntimechan`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `ntimechan`
integral_counts	*NX_INT*	*NX_UNITLESS*	

NXtofraw/NXentry/NXdata Members Declarations in the *NXtofraw/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
detector_number	*link*		target = `/NXentry/NXinstrument/NXdetecto`
time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXtofraw/NXentry/NXinstrument/NXdetector Members Declarations in the *NXtofraw/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=2) • 1: `ndet` • 2: `ntimechan`
detector_number	*NX_INT*		Dimensions: (rank=1) • 1: `ndet`
distance	*NX_FLOAT*	*NX_LENGTH*	distance to sample for each detector Dimensions: (rank=1) • 1: `ndet`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `ntimechan`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	polar angle for each detector element Dimensions: (rank=1) • 1: `ndet`
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	polar angle for each detector element Dimensions: (rank=1) • 1: `ndet`

NXtofsingle

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*, *NXuser*

This is a application definition for raw data from a generic TOF instrument

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXtofsingle.nxdl.xml

svnid: Id: NXtofsingle.nxdl.xml 1130 2012-09-19 16:02:54Z Freddie Akeroyd

Structure of NXtofsingle

```
NXtofsingle (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    duration:NX_FLOAT
    pre_sample_flightpath:NX_FLOAT
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      detector_number --> /NXentry/NXinstrument/NXdetector/detector_number
      time_of_flight --> /NXentry/NXinstrument/NXdetector/time_of_flight
    NXinstrument
      detector:NXdetector
        azimuthal_angle:NX_FLOAT[ndet]
        data:NX_INT[xsize,ysize,ntimechan]
        distance:NX_FLOAT[1]
        polar_angle:NX_FLOAT[ndet]
        time_of_flight:NX_FLOAT[ntimechan]
    NXmonitor
      data:NX_INT[ntimechan]
      distance:NX_FLOAT
      mode:NX_CHAR
      preset:NX_FLOAT
      time_of_flight:NX_FLOAT[ntimechan]
    NXsample
      name:NX_CHAR
      nature:NX_CHAR
    user:NXuser
      name:NX_CHAR
```

NXtofsingle Members Declarations in the *NXtofsingle* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXtofsingle/NXentry Members*.

NXtofsingle/NXentry Members Declarations in the *NXtofsingle/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXtofsingle`
duration	*NX_FLOAT*		
pre_sample_flightpath	*NX_FLOAT*	*NX_LENGTH*	This is the flight path before the sample position. This can be determined by a chopper, by the moderator or the source itself. In other words: it the distance to the component which gives the T0 signal to the detector electronics. If another component in the NXinstrument hierarchy provides this information, this should be a link.
user	*NXuser*		See table *NXtofsingle/NXentry/NXuser Members*.
(instrument)	*NXinstrument*		See table *NXtofsingle/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXtofsingle/NXentry/NXsample Members*.
(monitor)	*NXmonitor*		See table *NXtofsingle/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXtofsingle/NXentry/NXdata Members*.

NXtofsingle/NXentry/NXuser Members Declarations in the *NXtofsingle/NXentry/NXuser* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		

NXtofsingle/NXentry/NXinstrument Members Declarations in the *NXtofsingle/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		See table *NXtofsingle/NXentry/NXinstrument/NXdetector Members*.

NXtofsingle/NXentry/NXsample Members Declarations in the *NXtofsingle/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
nature	*NX_CHAR*		Any of these value(s): • `powder`: • `liquid`: • `single crystal`:

NXtofsingle/NXentry/NXmonitor Members Declarations in the *NXtofsingle/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
distance	*NX_FLOAT*	*NX_LENGTH*	
data	*NX_INT*		Dimensions: (rank=1) • 1: `ntimechan`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `ntimechan`

NXtofsingle/NXentry/NXdata Members Declarations in the *NXtofsingle/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
detector_number	*link*		target = `/NXentry/NXinstrument/NXdetecto`
time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXtofsingle/NXentry/NXinstrument/NXdetector Members Declarations in the *NXtofsingle/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=3) • 1: `xsize` • 2: `ysize` • 3: `ntimechan`
distance	*NX_FLOAT*	*NX_LENGTH*	Distance to sample for the center of the detector Dimensions: (rank=1) • 1: `1`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `ntimechan`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	polar angle for each detector element Dimensions: (rank=1) • 1: `ndet`
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	azimuthal angle for each detector element Dimensions: (rank=1) • 1: `ndet`

NXtomo

version	category	extends	groups cited
2.0	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*, *NXsource*

This is the application definition for x-ray or neutron tomography raw data. In tomography a number of dark field images are measured, some bright field images and, of course the sample. In order to distinguish between them images carry a image_key.

symbol list: These symbols will be used below to coordinate datasets with the same shape.

nFrames number of frames

xsize number of pixels in X direction

ysize number of pixels in Y direction

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXtomo.nxdl.xml

svnid: Id: NXtomo.nxdl.xml 1227 2013-08-29 15:33:28Z Tobias Richter

Structure of NXtomo

```
NXtomo (application definition, version 2.0)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    end_time:NX_DATE_TIME
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/detector:NXdetector/data
      image_key --> /NXentry/NXinstrument/detector:NXdetector/image_key
      rotation_angle --> /NXentry/NXsample/rotation_angle
    instrument:NXinstrument
      detector:NXdetector
        data:NX_INT[nFrames,xsize,ysize]
        distance:NX_FLOAT
        image_key:NX_INT[nFrames]
        x_pixel_size:NX_FLOAT
        x_rotation_axis_pixel_position:NX_FLOAT
        y_pixel_size:NX_FLOAT
        y_rotation_axis_pixel_position:NX_FLOAT
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    control:NXmonitor
      data:NX_FLOAT[nFrames]
    sample:NXsample
      name:NX_CHAR
      rotation_angle:NX_FLOAT[nFrames]
      x_translation:NX_FLOAT[nFrames]
      y_translation:NX_FLOAT[nFrames]
      z_translation:NX_FLOAT[nFrames]
```

NXtomo Members Declarations in the *NXtomo* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXtomo/NXentry Members*.

NXtomo/NXentry Members Declarations in the *NXtomo/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
end_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXtomo`
instrument	*NXinstrument*		See table *NXtomo/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXtomo/NXentry/NXsample Members*.
control	*NXmonitor*		See table *NXtomo/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXtomo/NXentry/NXdata Members*.

NXtomo/NXentry/NXinstrument Members Declarations in the *NXtomo/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXtomo/NXentry/NXinstrument/NXsource Members*.
detector	*NXdetector*		See table *NXtomo/NXentry/NXinstrument/NXdetector Members*.

NXtomo/NXentry/NXsample Members Declarations in the *NXtomo/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	In practice this axis is always aligned along one pixel direction on the detector and usually vertical. There are experiments with horizontal rotation axes, so this would need to be indicated somehow. For now the best way for that is an open question. Dimensions: (rank=1) • 1: `nFrames`
x_translation	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `nFrames`
y_translation	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `nFrames`
z_translation	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `nFrames`

NXtomo/NXentry/NXmonitor Members Declarations in the *NXtomo/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_FLOAT*	*NX_ANY*	Total integral monitor counts for each measured frame. Allows a to correction for fluctuations in the beam between frames Dimensions: (rank=1) • 1: `nFrames`

NXtomo/NXentry/NXdata Members Declarations in the *NXtomo/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/detector:`
rotation_angle	*link*		target = `/NXentry/NXsample/rotation_angl`
image_key	*link*		target = `/NXentry/NXinstrument/detector:`

NXtomo/NXentry/NXinstrument/NXsource Members Declarations in the *NXtomo/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXtomo/NXentry/NXinstrument/NXdetector Members Declarations in the *NXtomo/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=3) • 1: `nFrames` • 2: `xsize` • 3: `ysize`
image_key	*NX_INT*		In order to distinguish between sample projections, dark and flat images, a magic number is recorded per frame. The key is as follows: • projection = 0 • flat field = 1 • dark field = 2 • invalid = 3 Dimensions: (rank=1) • 1: `nFrames`
x_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
y_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
distance	*NX_FLOAT*	*NX_LENGTH*	Distance between detector and sample
x_rotation_axis_pixel_position	*NX_FLOAT*		
y_rotation_axis_pixel_position	*NX_FLOAT*		

NXtomophase

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXsample*, *NXsou*

This is the application definition for x-ray or neutron tomography raw data with phase contrast variation at each point. In tomography first some dark field images are measured, some bright field images and, of course the sample. In order to properly sort the order of the images taken, a sequence number is stored with each image.

symbol list: These symbols will be used below to coordinate datasets with the same shape.

nBrightFrames number of bright frames

nDarkFrames number of dark frames

nSampleFrames number of image (sample) frames

nPhase number of phase settings

xsize number of pixels in X direction

ysize number of pixels in Y direction

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXtomophase.nxdl.xml

svnid: Id: NXtomophase.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXtomophase

```
NXtomophase (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    end_time:NX_DATE_TIME
    start_time:NX_DATE_TIME
    title:NX_CHAR
    data:NXdata
      data --> /NXentry/NXinstrument/sample:NXdetector/data
      rotation_angle --> /NXentry/NXsample/rotation_angle
    instrument:NXinstrument
      bright_field:NXdetector
        data:NX_INT[nBrightFrames,xsize,ysize]
        sequence_number:NX_INT[nBrightFrames]
      dark_field:NXdetector
        data:NX_INT[nDarkFrames,xsize,ysize]
        sequence_number:NX_INT[nDarkFrames]
      sample:NXdetector
        data:NX_INT[nSampleFrames,nPhase,xsize,ysize]
        distance:NX_FLOAT
        sequence_number:NX_INT[nSampleFrames,nPhase]
        x_pixel_size:NX_FLOAT
        y_pixel_size:NX_FLOAT
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    control:NXmonitor
      integral:NX_FLOAT[nDarkFrames + nBrightFrames + nSampleFrame]
    sample:NXsample
      name:NX_CHAR
      rotation_angle:NX_FLOAT[nSampleFrames]
      x_translation:NX_FLOAT[nSampleFrames]
```

```
34        y_translation:NX_FLOAT[nSampleFrames]
35        z_translation:NX_FLOAT[nSampleFrames]
```

NXtomophase Members Declarations in the *NXtomophase* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXtomophase/NXentry Members*.

NXtomophase/NXentry Members Declarations in the *NXtomophase/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
end_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXtomophase`
instrument	*NXinstrument*		See table *NXtomophase/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXtomophase/NXentry/NXsample Members*.
control	*NXmonitor*		See table *NXtomophase/NXentry/NXmonitor Members*.
data	*NXdata*		See table *NXtomophase/NXentry/NXdata Members*.

NXtomophase/NXentry/NXinstrument Members Declarations in the *NXtomophase/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXtomophase/NXentry/NXinstrument/NXsource Members*.
bright_field	*NXdetector*		See table *NXtomophase/NXentry/NXinstrument/NXdetector Members*.
dark_field	*NXdetector*		See table *NXtomophase/NXentry/NXinstrument/NXdetector Members*.
sample	*NXdetector*		See table *NXtomophase/NXentry/NXinstrument/NXdetector Members*.

NXtomophase/NXentry/NXsample Members Declarations in the *NXtomophase/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=1) • 1: `nSampleFrames`
x_translation	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `nSampleFrames`
y_translation	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `nSampleFrames`
z_translation	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `nSampleFrames`

NXtomophase/NXentry/NXmonitor Members Declarations in the *NXtomophase/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
integral	*NX_FLOAT*	*NX_ANY*	Total integral monitor counts for each measured frame. Allows a correction for fluctuations in the beam between frames Dimensions: (rank=1) • 1: `nDarkFrames + nBrightFrames + nSampleFrame`

NXtomophase/NXentry/NXdata Members Declarations in the *NXtomophase/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/sample:NX`
rotation_angle	*link*		target = `/NXentry/NXsample/rotation_angl`

NXtomophase/NXentry/NXinstrument/NXsource Members Declarations in the *NXtomophase/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXtomophase/NXentry/NXinstrument/NXdetector Members Declarations in the *NXtomophase/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=3) • 1: `nBrightFrames` • 2: `xsize` • 3: `ysize`
sequence_number	*NX_INT*		Dimensions: (rank=1) • 1: `nBrightFrames`

NXtomophase/NXentry/NXinstrument/NXdetector Members Declarations in the *NXtomophase/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=3) • 1: `nDarkFrames` • 2: `xsize` • 3: `ysize`
sequence_number	*NX_INT*		Dimensions: (rank=1) • 1: `nDarkFrames`

NXtomophase/NXentry/NXinstrument/NXdetector Members Declarations in the *NXtomophase/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=4) • 1: `nSampleFrames` • 2: `nPhase` • 3: `xsize` • 4: `ysize`
sequence_number	*NX_INT*		Dimensions: (rank=2) • 1: `nSampleFrames` • 2: `nPhase`
x_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
y_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
distance	*NX_FLOAT*	*NX_LENGTH*	Distance between detector and sample

NXtomoproc

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXentry*, *NXinstrument*, *NXparameters*, *NXprocess*, *NXsample*, *NXso*

This is an application definition for the final result of a tomography experiment: a 3D construction of some volume of physical properties.

symbol list: These symbols will be used below to coordinate datasets with the same shape.

nx number of voxels in X direction

ny number of voxels in Y direction

nz number of voxels in Z direction

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXtomoproc.nxdl.xml

svnid: Id: NXtomoproc.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXtomoproc

```
NXtomoproc (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    title:NX_CHAR
    data:NXdata
      data:NX_INT[nx,nx,nz]
        @transform
        @offset
        @scaling
      x:NX_FLOAT[nx]
      y:NX_FLOAT[ny]
      z:NX_FLOAT[nz]
    NXinstrument
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    reconstruction:NXprocess
      date:NX_DATE_TIME
      program:NX_CHAR
      version:NX_CHAR
      parameters:NXparameters
        raw_file:NX_CHAR
    NXsample
      name:NX_CHAR
```

NXtomoproc Members Declarations in the *NXtomoproc* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXtomoproc/NXentry Members*.

NXtomoproc/NXentry Members Declarations in the *NXtomoproc/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXtomoproc`
(instrument)	*NXinstrument*		See table *NXtomoproc/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXtomoproc/NXentry/NXsample Members*.
reconstruction	*NXprocess*		See table *NXtomoproc/NXentry/NXprocess Members*.
data	*NXdata*		See table *NXtomoproc/NXentry/NXdata Members*.

NXtomoproc/NXentry/NXinstrument Members Declarations in the *NXtomoproc/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXtomoproc/NXentry/NXinstrument/NXsource Members*.

NXtomoproc/NXentry/NXsample Members Declarations in the *NXtomoproc/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample

NXtomoproc/NXentry/NXprocess Members Declarations in the *NXtomoproc/NXentry/NXprocess* group.

Name and Attributes	Type	Units	Description (and Occurrences)
program	*NX_CHAR*		Name of the program used for reconstruction
version	*NX_CHAR*		Version of the program used
date	*NX_DATE_TIME*		Date and time of reconstruction processing.
parameters	*NXparameters*		See table *NXtomoproc/NXentry/NXprocess/NXparameters Members*.

NXtomoproc/NXentry/NXdata Members Declarations in the *NXtomoproc/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		This is the reconstructed volume. This can be different things. Please indicate in the unit attribute what physical quantity this really is. Dimensions: (rank=3) • 1: `nx` • 2: `nx` • 3: `nz`
@transform	(*NX_CHAR*)		
@offset	(*NX_CHAR*)		
@scaling	(*NX_CHAR*)		
x	*NX_FLOAT*	*NX_ANY*	This is an array holding the values to use for the x-axis of data. The units must be appropriate for the measurement. Dimensions: (rank=1) • 1: `nx`
y	*NX_FLOAT*	*NX_ANY*	This is an array holding the values to use for the y-axis of data. The units must be appropriate for the measurement. Dimensions: (rank=1) • 1: `ny`
z	*NX_FLOAT*	*NX_ANY*	This is an array holding the values to use for the z-axis of data. The units must be appropriate for the measurement. Dimensions: (rank=1) • 1: `nz`

NXtomoproc/NXentry/NXinstrument/NXsource **Members** Declarations in the *NXtomoproc/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXtomoproc/NXentry/NXprocess/NXparameters **Members** Declarations in the *NXtomoproc/NXentry/NXprocess/NXparameters* group.

Name and Attributes	Type	Units	Description (and Occurrences)
raw_file	*NX_CHAR*		Original raw data file this data was derived from

NXxas

version	category	extends	groups cited
1.0	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXmonochromator*, *NXsample*, *NXsource*

This is an application definition for raw data from an X-ray absorption spectroscopy experiment. This is essentially a scan on energy versus incoming/ absorbed beam

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxas.nxdl.xml

svnid: Id: NXxas.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXxas

```
NXxas (application definition, version 1.0)
  (overlays NXentry)
  NXentry
    @entry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    NXdata
      absorbed_beam --> /entry/instrument/absorbed_beam/data
      energy --> /entry/instrument/monochromator/energy
    NXinstrument
      incoming_beam:NXdetector
        data:NX_INT[np]
      absorbed_beam:NXdetector
        data:NX_INT[np]
      monochromator:NXmonochromator
        energy:NX_FLOAT[np]
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
```

```
    NXmonitor
      data:NX_INT[np]
      mode:NX_CHAR
      preset:NX_FLOAT
    NXsample
      name:NX_CHAR
```

NXxas Members Declarations in the *NXxas* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXxas/NXentry Members*.
@entry	(*NX_CHAR*)		NeXus convention is to use "entry1", "entry2", ... for analysis software to locate each entry

NXxas/NXentry Members Declarations in the *NXxas/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxas`
(instrument)	*NXinstrument*		See table *NXxas/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXxas/NXentry/NXsample Members*.
(monitor)	*NXmonitor*		See table *NXxas/NXentry/NXmonitor Members*.
(data)	*NXdata*		See table *NXxas/NXentry/NXdata Members*.

NXxas/NXentry/NXinstrument Members Declarations in the *NXxas/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXxas/NXentry/NXinstrument/NXsource Members*.
monochromator	*NXmonochromator*		See table *NXxas/NXentry/NXinstrument/NXmonochromator Members*.
incoming_beam	*NXdetector*		See table *NXxas/NXentry/NXinstrument/NXdetector Members*.
absorbed_beam	*NXdetector*		See table *NXxas/NXentry/NXinstrument/NXdetector Members*.

NXxas/NXentry/NXsample Members Declarations in the *NXxas/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample

NXxas/NXentry/NXmonitor Members Declarations in the *NXxas/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
data	*NX_INT*		Dimensions: (rank=1) • 1: `np`

NXxas/NXentry/NXdata Members Declarations in the *NXxas/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
energy	*link*		target = `/entry/instrument/monochromator`
absorbed_beam	*link*		target = `/entry/instrument/absorbed_beam`

NXxas/NXentry/NXinstrument/NXsource Members Declarations in the *NXxas/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		This value: `x-ray`

NXxas/NXentry/NXinstrument/NXmonochromator Members Declarations in the *NXxas/NXentry/NXinstrument/NXmonochromator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
energy	*NX_FLOAT*		Dimensions: (rank=1) • 1: `np`

NXxas/NXentry/NXinstrument/NXdetector Members Declarations in the *NXxas/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		Dimensions: (rank=1) • 1: `np`

NXxas/NXentry/NXinstrument/NXdetector Members Declarations in the *NXxas/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		mark this field with attribute `signal=1` Dimensions: (rank=1) • 1: `np`

NXxasproc	version	category	extends	groups cited
	1.0	application	*NXobject*	*NXdata*, *NXentry*, *NXparameters*, *NXprocess*, *NXsample*

This is an application definition for processed data from XAS. This is energy versus I(incoming)/I(absorbed)

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxasproc.nxdl.xml

svnid: Id: NXxasproc.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXxasproc

```
NXxasproc (application definition, version 1.0)
  (overlays NXentry)
  NXentry
    @entry
    definition:NX_CHAR
    title:NX_CHAR
    NXdata
      data:NX_FLOAT[np]
      energy:NX_CHAR
    XAS_data_reduction:NXprocess
      date:NX_DATE_TIME
      program:NX_CHAR
      version:NX_CHAR
      parameters:NXparameters
        raw_file:NX_CHAR
    NXsample
      name:NX_CHAR
```

NXxasproc Members Declarations in the *NXxasproc* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXxasproc/NXentry Members*.
@entry	(*NX_CHAR*)		NeXus convention is to use "entry1", "entry2", ... for analysis software to locate each entry

NXxasproc/NXentry Members Declarations in the *NXxasproc/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxasproc`
(sample)	*NXsample*		See table *NXxasproc/NXentry/NXsample Members*.
XAS_data_reduction	*NXprocess*		See table *NXxasproc/NXentry/NXprocess Members*.
(data)	*NXdata*		See table *NXxasproc/NXentry/NXdata Members*.

NXxasproc/NXentry/NXsample Members Declarations in the *NXxasproc/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	(*NX_CHAR*)		Descriptive name of sample

NXxasproc/NXentry/NXprocess Members Declarations in the *NXxasproc/NXentry/NXprocess* group.

Name and Attributes	Type	Units	Description (and Occurrences)
program	*NX_CHAR*		Name of the program used for reconstruction
version	*NX_CHAR*		Version of the program used
date	*NX_DATE_TIME*		Date and time of reconstruction processing.
parameters	*NXparameters*		See table *NXxasproc/NXentry/NXprocess/NXparameters Members*.

NXxasproc/NXentry/NXdata Members Declarations in the *NXxasproc/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
energy	(*NX_CHAR*)		Dimensions: (rank=1) • 1: `np`
data	*NX_FLOAT*		This is corrected and calibrated I(incoming)/I(absorbed). So it is the absorption. Expect attribute `signal=1` Dimensions: (rank=1) • 1: `np`

NXxasproc/NXentry/NXprocess/NXparameters Members Declarations in the *NXxasproc/NXentry/NXprocess/NXparameters* group.

Name and Attributes	Type	Units	Description (and Occurrences)
raw_file	*NX_CHAR*		Original raw data file this data was derived from

NXxbase

version	category	extends	groups cited
1.0b	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonitor*, *NXmonochromator*, *NXsample*, *NXsource*

This definition covers the common parts of all monochromatic single crystal raw data application definitions

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxbase.nxdl.xml

svnid: Id: NXxbase.nxdl.xml 1130 2012-09-19 16:02:54Z Freddie Akeroyd

Structure of NXxbase

```
NXxbase (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
    instrument:NXinstrument
      detector:NXdetector
```

```
        data:NX_INT[np,number of x pixels,number of y pixels]
          @signal
        distance:NX_FLOAT
        frame_start_number:NX_INT
        x_pixel_size:NX_FLOAT
        y_pixel_size:NX_FLOAT
      monochromator:NXmonochromator
        wavelength:NX_FLOAT
      source:NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    control:NXmonitor
      integral:NX_FLOAT
      mode:NX_CHAR
      preset:NX_FLOAT
    sample:NXsample
      distance:NX_FLOAT
      name:NX_CHAR
      orientation_matrix:NX_FLOAT[3,3]
      temperature:NX_FLOAT[NP]
      unit_cell:NX_FLOAT[6]
      x_translation:NX_FLOAT
      y_translation:NX_FLOAT
```

NXxbase Members Declarations in the *NXxbase* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXxbase/NXentry Members*.

NXxbase/NXentry Members Declarations in the *NXxbase/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxbase`
instrument	*NXinstrument*		See table *NXxbase/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXxbase/NXentry/NXsample Members*.
control	*NXmonitor*		See table *NXxbase/NXentry/NXmonitor Members*.
(data)	*NXdata*		The name of this group id data if there is only one detector; if there are several the names will be data1, data2, data3 and will point to the corresponding detector groups in the instrument hierarchy. See table *NXxbase/NXentry/NXdata Members*.

NXxbase/NXentry/NXinstrument Members Declarations in the *NXxbase/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
source	*NXsource*		See table *NXxbase/NXentry/NXinstrument/NXsource Members*.
monochromator	*NXmonochromator*		See table *NXxbase/NXentry/NXinstrument/NXmonochr Members*.
detector	*NXdetector*		The name of the group is detector if there is only one detector, if there are several, names have to be detector1, detector2, ...detectorn See table *NXxbase/NXentry/NXinstrument/NXdetector Members*.

NXxbase/NXentry/NXsample Members Declarations in the *NXxbase/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		Descriptive name of sample
orientation_matrix	*NX_FLOAT*		The orientation matrix according to Busing and Levy conventions. This is not strictly necessary as the UB can always be derived from the data. But let us bow to common usage which includes the UB nearly always. Dimensions: (rank=2) • 1: `3` • 2: `3`
unit_cell	*NX_FLOAT*		The unit cell, a, b, c, alpha, beta, gamma. Again, not strictly necessary, but normally written. Dimensions: (rank=1) • 1: `6`
temperature	*NX_FLOAT*		The sample temperature or whatever sensor represents this value best Dimensions: (rank=1) • 1: `NP`
x_translation	*NX_FLOAT*	*NX_LENGTH*	Translation of the sample along the X-direction of the laboratory coordinate system
y_translation	*NX_FLOAT*	*NX_LENGTH*	Translation of the sample along the Y-direction of the laboratory coordinate system
distance	*NX_FLOAT*	*NX_LENGTH*	Translation of the sample along the Z-direction of the laboratory coordinate system

NXxbase/NXentry/NXmonitor Members Declarations in the *NXxbase/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
mode	(*NX_CHAR*)		Count to a preset value based on either clock time (timer) or received monitor counts (monitor). Any of these value(s): • `monitor`: • `timer`:
preset	*NX_FLOAT*		preset value for time or monitor
integral	*NX_FLOAT*	*NX_ANY*	Total integral monitor counts

NXxbase/NXentry/NXdata Members Declarations in the *NXxbase/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXxbase/NXentry/NXinstrument/NXsource Members Declarations in the *NXxbase/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		Any of these value(s): • `neutron`: • `x-ray`: • `electron`:

NXxbase/NXentry/NXinstrument/NXmonochromator Members Declarations in the *NXxbase/NXentry/NXinstrument/NXmonochromator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	

NXxbase/NXentry/NXinstrument/NXdetector Members Declarations in the *NXxbase/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_INT*		The area detector data, the first dimension is always the number of scan points, the second and third are the number of pixels in x and y. The origin is always assumed to be in the center of the detector. maxOccurs is limited to the the number of detectors on your instrument Dimensions: (rank=3) • 1: `np` • 2: `number of x pixels` • 3: `number of y pixels`
@signal	*NX_POSINT*		This value: `1`
x_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
y_pixel_size	*NX_FLOAT*	*NX_LENGTH*	
distance	*NX_FLOAT*	*NX_LENGTH*	
frame_start_number	*NX_INT*		This is the start number of the first frame of a scan. In PX one often scans a couple of frames on a give sample, then does something else, then returns to the same sample and scans some more frames. Each time with a new data file. This number helps concatenating such measurements.

NXxeuler

version	category	extends	groups cited
1.0b	application	*NXxbase*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXsample*

This is the application definition for raw data from a four-circle diffractometer with an eulerian cradle. It extends *NXxbase*, so the full definition is the content of NXxbase plus the data defined here. All four angles are logged in order to support arbitrary scans in reciprocal space.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxeuler.nxdl.xml

svnid: Id: NXxeuler.nxdl.xml 1189 2012-10-25 13:25:37Z Pete Jemian

Structure of NXxeuler

```
NXxeuler (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    name:NXdata
      chi --> /NXentry/NXsample/chi
      phi --> /NXentry/NXsample/phi
      polar_angle --> /NXentry/NXinstrument/NXdetector/polar_angle
      rotation_angle --> /NXentry/NXsample/rotation_angle
    instrument:NXinstrument
      detector:NXdetector
        polar_angle:NX_FLOAT[np]
    sample:NXsample
      chi:NX_FLOAT[np]
      phi:NX_FLOAT[np]
      rotation_angle:NX_FLOAT[np]
```

NXxeuler Members Declarations in the *NXxeuler* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXxeuler/NXentry Members*.

NXxeuler/NXentry Members Declarations in the *NXxeuler/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxeuler`
instrument	*NXinstrument*		See table *NXxeuler/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXxeuler/NXentry/NXsample Members*.
name	*NXdata*		See table *NXxeuler/NXentry/NXdata Members*.

NXxeuler/NXentry/NXinstrument Members Declarations in the *NXxeuler/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		See table *NXxeuler/NXentry/NXinstrument/NXdetector Members*.

NXxeuler/NXentry/NXsample Members Declarations in the *NXxeuler/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the sample rotation angle at each scan point Dimensions: (rank=1) • 1: `np`
chi	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the chi angle of the eulerian cradle at each scan point Dimensions: (rank=1) • 1: `np`
phi	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the phi rotation of the eulerian cradle at each scan point Dimensions: (rank=1) • 1: `np`

NXxeuler/NXentry/NXdata Members Declarations in the *NXxeuler/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*link*		target = `/NXentry/NXinstrument/NXdetecto`
rotation_angle	*link*		target = `/NXentry/NXsample/rotation_angl`
chi	*link*		target = `/NXentry/NXsample/chi`
phi	*link*		target = `/NXentry/NXsample/phi`

NXxeuler/NXentry/NXinstrument/NXdetector Members Declarations in the *NXxeuler/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*NX_FLOAT*	*NX_ANGLE*	The polar_angle (two theta) where the detector is placed at each scan point. Dimensions: (rank=1) • 1: np

NXxkappa

version	category	extends	groups cited
1.0b	application	*NXxbase*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXsample*

This is the application definition for raw data from a kappa geometry (CAD4) single crystal diffractometer. It extends NXxbase, so the full definition is the content of NXxbase plus the data defined here.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxkappa.nxdl.xml

svnid: Id: NXxkappa.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXxkappa

```
NXxkappa (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    name:NXdata
      kappa --> /NXentry/NXsample/kappa
      phi --> /NXentry/NXsample/phi
      polar_angle --> /NXentry/NXinstrument/NXdetector/polar_angle
      rotation_angle --> /NXentry/NXsample/rotation_angle
    instrument:NXinstrument
      detector:NXdetector
        polar_angle:NX_FLOAT[np]
    sample:NXsample
      alpha:NX_FLOAT
      kappa:NX_FLOAT[np]
      phi:NX_FLOAT[np]
      rotation_angle:NX_FLOAT[np]
```

NXxkappa Members Declarations in the *NXxkappa* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXxkappa/NXentry Members*.

NXxkappa/NXentry Members Declarations in the *NXxkappa/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxkappa`
instrument	*NXinstrument*		See table *NXxkappa/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXxkappa/NXentry/NXsample Members*.
name	*NXdata*		See table *NXxkappa/NXentry/NXdata Members*.

NXxkappa/NXentry/NXinstrument Members Declarations in the *NXxkappa/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		See table *NXxkappa/NXentry/NXinstrument/NXdetecto Members*.

NXxkappa/NXentry/NXsample Members Declarations in the *NXxkappa/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the sample rotation angle at each scan point Dimensions: (rank=1) • 1: `np`
kappa	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the kappa angle at each scan point Dimensions: (rank=1) • 1: `np`
phi	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the phi angle at each scan point Dimensions: (rank=1) • 1: `np`
alpha	*NX_FLOAT*	*NX_ANGLE*	This holds the inclination angle of the kappa arm.

NXxkappa/NXentry/NXdata Members Declarations in the *NXxkappa/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*link*		target = `/NXentry/NXinstrument/NXdetecto`
rotation_angle	*link*		target = `/NXentry/NXsample/rotation_angl`
kappa	*link*		target = `/NXentry/NXsample/kappa`
phi	*link*		target = `/NXentry/NXsample/phi`

NXxkappa/NXentry/NXinstrument/NXdetector Members Declarations in the *NXxkappa/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*NX_FLOAT*	*NX_ANGLE*	The polar_angle (two theta) at each scan point Dimensions: (rank=1) • 1: `np`

NXxlaue

version	category	extends	groups cited
1.0b	application	*NXxrot*	*NXdata*, *NXentry*, *NXinstrument*, *NXsource*

This is the application definition for raw data from a single crystal laue camera. It extends NXxrot.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxlaue.nxdl.xml

svnid: Id: NXxlaue.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXxlaue

```
NXxlaue (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    instrument:NXinstrument
      source:NXsource
        distribution:NXdata
          data:NX_CHAR
          wavelength:NX_CHAR
```

NXxlaue Members Declarations in the *NXxlaue* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXxlaue/NXentry Members*.

NXxlaue/NXentry Members Declarations in the *NXxlaue/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxlaue`
instrument	*NXinstrument*		See table *NXxlaue/NXentry/NXinstrument Members*.

NXxlaue/NXentry/NXinstrument Members Declarations in the *NXxlaue/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
source	*NXsource*		See table *NXxlaue/NXentry/NXinstrument/NXsource Members*.

NXxlaue/NXentry/NXinstrument/NXsource Members Declarations in the *NXxlaue/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distribution	*NXdata*		This is the wavelength distribution of the beam See table *NXxlaue/NXentry/NXinstrument/NXsource/N Members*.

NXxlaue/NXentry/NXinstrument/NXsource/NXdata Members Declarations in the *NXxlaue/NXentry/NXinstrument/NXsource/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	(*NX_CHAR*)		expect `signal=1 axes="energy"` Dimensions: (rank=1) • 1: `ne`
wavelength	(*NX_CHAR*)	*NX_WAVELENGTH*	Dimensions: (rank=1) • 1: `ne`

NXxlaueplate

version	category	extends	groups cited
1.0b	application	*NXxlaue*	*NXdetector*, *NXentry*, *NXinstrument*

This is the application definition for raw data from a single crystal Laue camera with an image plate as a detector. It extends NXxlaue.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxlaueplate.nxdl.xml

svnid: Id: NXxlaueplate.nxdl.xml 1130 2012-09-19 16:02:54Z Freddie Akeroyd

Structure of NXxlaueplate

```
NXxlaueplate (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    instrument:NXinstrument
      detector:NXdetector
        diameter:NX_FLOAT
```

NXxlaueplate Members Declarations in the *NXxlaueplate* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXxlaueplate/NXentry Members*.

NXxlaueplate/NXentry Members Declarations in the *NXxlaueplate/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxlaueplate`
instrument	*NXinstrument*		See table *NXxlaueplate/NXentry/NXinstrument Members*.

NXxlaueplate/NXentry/NXinstrument Members Declarations in the *NXxlaueplate/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		See table *NXxlaueplate/NXentry/NXinstrument/NXdetector Members*.

NXxlaueplate/NXentry/NXinstrument/NXdetector Members Declarations in the *NXxlaueplate/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
diameter	*NX_FLOAT*	*NX_LENGTH*	The diameter of a cylindrical detector

NXxnb

version	category	extends	groups cited
1.0b	application	*NXxbase*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXsample*

This is the application definition for raw data from a single crystal diffractometer measuring in normal beam mode. It extends NXxbase, so the full definition is the content of NXxbase plus the data defined here. All angles are logged in order to support arbitrary scans in reciprocal space.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxnb.nxdl.xml

svnid: Id: NXxnb.nxdl.xml 1130 2012-09-19 16:02:54Z Freddie Akeroyd

Structure of NXxnb

```
NXxnb (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    name:NXdata
      polar_angle --> /NXentry/NXinstrument/NXdetector/polar_angle
      rotation_angle --> /NXentry/NXsample/rotation_angle
      tilt --> /NXentry/NXinstrument/NXdetector/tilt
    instrument:NXinstrument
      detector:NXdetector
        polar_angle:NX_FLOAT[np]
        tilt_angle:NX_FLOAT[np]
    sample:NXsample
      rotation_angle:NX_FLOAT[np]
```

NXxnb Members Declarations in the *NXxnb* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXxnb/NXentry Members*.

NXxnb/NXentry Members Declarations in the *NXxnb/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxnb`
instrument	*NXinstrument*		See table *NXxnb/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXxnb/NXentry/NXsample Members*.
name	*NXdata*		See table *NXxnb/NXentry/NXdata Members*.

NXxnb/NXentry/NXinstrument Members Declarations in the *NXxnb/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		See table *NXxnb/NXentry/NXinstrument/NXdetector Members*.

NXxnb/NXentry/NXsample Members Declarations in the *NXxnb/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the sample rotation angle at each scan point Dimensions: (rank=1) • 1: `np`

NXxnb/NXentry/NXdata Members Declarations in the *NXxnb/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*link*		target = `/NXentry/NXinstrument/NXdetecto`
tilt	*link*		target = `/NXentry/NXinstrument/NXdetecto`
rotation_angle	*link*		target = `/NXentry/NXsample/rotation_angl`

NXxnb/NXentry/NXinstrument/NXdetector Members Declarations in the *NXxnb/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*NX_FLOAT*	*NX_ANGLE*	The polar_angle (gamma) of the detector for each scan point. Dimensions: (rank=1) • 1: `np`
tilt_angle	*NX_FLOAT*	*NX_ANGLE*	The angle by which the detector has been tilted out of the scattering plane. Dimensions: (rank=1) • 1: `np`

NXxrot

version	category	extends	groups cited
1.0b	application	*NXxbase*	*NXattenuator*, *NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXsample*

This is the application definition for raw data from a rotation camera. It extends NXxbase, so the full definition is the content of NXxbase plus the data defined here.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/applications/NXxrot.nxdl.xml

svnid: Id: NXxrot.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXxrot

```
NXxrot (application definition, version 1.0b)
  (overlays NXentry)
  entry:NXentry
    definition:NX_CHAR
    name:NXdata
      rotation_angle --> /NXentry/NXsample/rotation_angle
    instrument:NXinstrument
      attenuator:NXattenuator
        attenuator_transmission:NX_FLOAT
      detector:NXdetector
        beam_center_x:NX_FLOAT
        beam_center_y:NX_FLOAT
        polar_angle:NX_FLOAT
    sample:NXsample
      rotation_angle:NX_FLOAT[np]
      rotation_angle_step:NX_FLOAT[np]
```

NXxrot Members Declarations in the *NXxrot* group.

Name and Attributes	Type	Units	Description (and Occurrences)
entry	*NXentry*		See table *NXxrot/NXentry Members*.

NXxrot/NXentry Members Declarations in the *NXxrot/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXxrot`
instrument	*NXinstrument*		See table *NXxrot/NXentry/NXinstrument Members*.
sample	*NXsample*		See table *NXxrot/NXentry/NXsample Members*.
name	*NXdata*		See table *NXxrot/NXentry/NXdata Members*.

NXxrot/NXentry/NXinstrument Members Declarations in the *NXxrot/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
detector	*NXdetector*		See table *NXxrot/NXentry/NXinstrument/NXdetector Members*.
attenuator	*NXattenuator*		See table *NXxrot/NXentry/NXinstrument/NXattenuator Members*.

NXxrot/NXentry/NXsample Members Declarations in the *NXxrot/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_angle	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the sample rotation start angle at each scan point Dimensions: (rank=1) • 1: `np`
rotation_angle_step	*NX_FLOAT*	*NX_ANGLE*	This is an array holding the step made for sample rotation angle at each scan point Dimensions: (rank=1) • 1: `np`

NXxrot/NXentry/NXdata Members Declarations in the *NXxrot/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
rotation_angle	*link*		target = `/NXentry/NXsample/rotation_angl`

NXxrot/NXentry/NXinstrument/NXdetector Members Declarations in the *NXxrot/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
polar_angle	*NX_FLOAT*	*NX_ANGLE*	The polar_angle (two theta) where the detector is placed.
beam_center_x	*NX_FLOAT*	*NX_LENGTH*	This is the x position where the direct beam would hit the detector. This is a length, not a pixel position, and can be outside of the actual detector.
beam_center_y	*NX_FLOAT*	*NX_LENGTH*	This is the y position where the direct beam would hit the detector. This is a length, not a pixel position, and can be outside of the actual detector.

NXxrot/NXentry/NXinstrument/NXattenuator Members Declarations in the *NXxrot/NXentry/NXinstrument/NXattenuator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
attenuator_transmission	*NX_FLOAT*	*NX_ANY*	

Contributed Definitions

A description of each NeXus contributed definition is given. NXDL files in the NeXus contributed definitions include propositions from the community for NeXus base classes or application definitions, as well as other NXDL files for long-term archival by NeXus. Consider the contributed definitions as either in *incubation* or a special case not for general use. The *NIAC: The NeXus International Advisory Committee* is charged to review any new contributed definitions and provide feedback to the authors before ratification and acceptance as either a base class or application definition.

NXarpes

version	category	extends	groups cited
1.0	application	*NXobject*	*NXdata*, *NXdetector*, *NXentry*, *NXinstrument*, *NXmonochromator*, *NXsample*, *NXsource*

This is an application definition for angular resolved photo electron spectroscopy. It has been drawn up with hemispherical electron analysers in mind.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXarpes.nxdl.xml

svnid: Id: NXarpes.nxdl.xml 1236 2013-09-16 02:37:59Z Pete Jemian

Structure of NXarpes

```
NXarpes (application definition, version 1.0)
  (overlays NXentry)
  NXentry
    @entry
    definition:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    NXdata
    NXinstrument
      analyser:NXdetector
        acquisition_mode:NX_CHAR
        angles:NX_NUMBER
        data:NX_NUMBER
        energies:NX_NUMBER
        entrance_slit_setting:NX_NUMBER
        entrance_slit_shape:NX_CHAR
        entrance_slit_size:NX_CHAR
        lens_mode:NX_CHAR
        pass_energy:NX_CHAR
        region_origin:NX_INT[]
        region_size:NX_INT[]
        sensor_size:NX_INT[]
        time_per_channel:NX_CHAR
      monochromator:NXmonochromator
        energy:NX_NUMBER
      NXsource
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    NXsample
      name:NX_CHAR
      temperature:NX_CHAR
```

NXarpes Members Declarations in the *NXarpes* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXarpes/NXentry Members*.
@entry	(*NX_CHAR*)		NeXus convention is to use “entry1”, “entry2”, ... for analysis software to locate each entry

NXarpes/NXentry Members Declarations in the *NXarpes/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
title	*NX_CHAR*		
start_time	*NX_DATE_TIME*		
definition	(*NX_CHAR*)		Official NeXus NXDL schema to which this file conforms This value: `NXarpes`
(instrument)	*NXinstrument*		See table *NXarpes/NXentry/NXinstrument Members*.
(sample)	*NXsample*		See table *NXarpes/NXentry/NXsample Members*.
(data)	*NXdata*		

NXarpes/NXentry/NXinstrument Members Declarations in the *NXarpes/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(source)	*NXsource*		See table *NXarpes/NXentry/NXinstrument/NXsource Members*.
monochromator	*NXmonochromator*		See table *NXarpes/NXentry/NXinstrument/NXmonochr Members*.
analyser	*NXdetector*		See table *NXarpes/NXentry/NXinstrument/NXdetector Members*.

NXarpes/NXentry/NXsample Members Declarations in the *NXarpes/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
name	*NX_CHAR*		Descriptive name of sample
temperature	(*NX_CHAR*)	*NX_TEMPERATURE*	

NXarpes/NXentry/NXinstrument/NXsource Members Declarations in the *NXarpes/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	*NX_CHAR*		
name	*NX_CHAR*		
probe	(*NX_CHAR*)		This value: `x-ray`

NXarpes/NXentry/NXinstrument/NXmonochromator Members Declarations in the *NXarpes/NXentry/NXinstrument/NXmonochromator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
energy	*NX_NUMBER*	*NX_ENERGY*	

NXarpes/NXentry/NXinstrument/NXdetector Members Declarations in the *NXarpes/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_NUMBER*		
lens_mode	*NX_CHAR*		setting for the electron analyser lens
acquisition_mode	(*NX_CHAR*)		Any of these value(s): • `swept`: • `fixed`:
entrance_slit_shape	(*NX_CHAR*)		Any of these value(s): • `curved`: • `straight`:
entrance_slit_setting	*NX_NUMBER*	*NX_ANY*	dial setting of the entrance slit
entrance_slit_size	(*NX_CHAR*)	*NX_LENGTH*	size of the entrance slit
pass_energy	(*NX_CHAR*)	*NX_ENERGY*	energy of the electrons on the mean path of the analyser
time_per_channel	(*NX_CHAR*)	*NX_TIME*	todo: define more clearly
angles	*NX_NUMBER*	*NX_ANGLE*	angular axis of the analyser data which dimension the axis applies to is defined using the normal NXdata methods.
energies	*NX_NUMBER*	*NX_ENERGY*	energy axis of the analyser data which dimension the axis applies to is defined using the normal NXdata methods.
sensor_size	*NX_INT*		number of raw active elements in fast and slow pixel dimension direction
region_origin	*NX_INT*		origin of recangular region selected for readout
region_size	*NX_INT*		size of recangular region selected for readout

NXbeamline

version	category	extends	groups cited
1.0	contributed	*NXobject*	*NXaperture*, *NXbending_magnet*, *NXcollection*, *NXelectrostatic_kicker*, *NXmagnetic_kicker*, *NXquadrupole_magnet*, *NXseparator*, *NXsolenoid_magnet*, *NXspin_rotator*

container for elements describing beamline.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXbeamline.nxdl.xml

svnid: Id: NXbeamline.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXbeamline

```
NXbeamline (contributed definition, version 1.0)
  (base class definition, NXentry or NXsubentry not found)
  beamline:NX_CHAR
  NXaperture
  NXbending_magnet
  diagnostics:NXcollection
  NXelectrostatic_kicker
  NXmagnetic_kicker
  NXquadrupole_magnet
  NXseparator
  NXsolenoid_magnet
  NXspin_rotator
```

NXbeamline Members Declarations in the *NXbeamline* group.

Name and Attributes	Type	Units	Description (and Occurrences)
beamline	*NX_CHAR*		name of beamline.
diagnostics	*NXcollection*		
(bending_magnet)	*NXbending_magnet*		
(quadrupole_magnet)	*NXquadrupole_magnet*		
(solenoid_magnet)	*NXsolenoid_magnet*		
(separator)	*NXseparator*		
(spin_rotator)	*NXspin_rotator*		
(electrostatic_kicker)	*NXelectrostatic_kicker*		
(magnetic_kicker)	*NXmagnetic_kicker*		
(aperture)	*NXaperture*		

NXelectrostatic_kicker

version	category	extends	groups cited
1.0	contributed	*NXobject*	*NXlog*

definition for a electrostatic kicker.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXelectrostatic_kicker.nxdl.xml

svnid: Id: NXelectrostatic_kicker.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXelectrostatic_kicker

```
NXelectrostatic_kicker (contributed definition, version 1.0)
  (base class definition, NXentry or NXsubentry not found)
  beamline_distance:NX_FLOAT
  description:NX_CHAR
  set_current:NX_FLOAT
  set_voltage:NX_FLOAT
  timing:NX_FLOAT
    @description
  read_current:NXlog
    value:NX_CHAR
  read_voltage:NXlog
    value:NX_CHAR
```

NXelectrostatic_kicker Members Declarations in the *NXelectrostatic_kicker* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	*NX_CHAR*		extended description of the kicker.
beamline_distance	*NX_FLOAT*	*NX_LENGTH*	define position of beamline element relative to production target
timing	*NX_FLOAT*	*NX_TIME*	kicker timing as defined by ‘description’ attribute
@description	*NX_CHAR*		
set_current	*NX_FLOAT*	*NX_CURRENT*	current set on supply.
set_voltage	*NX_FLOAT*	*NX_VOLTAGE*	volage set on supply.
read_current	*NXlog*		current read from supply. See table *NXelectrostatic_kicker/NXlog Members*.
read_voltage	*NXlog*		voltage read from supply. See table *NXelectrostatic_kicker/NXlog Members*.

NXelectrostatic_kicker/NXlog Members Declarations in the *NXelectrostatic_kicker/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_CURRENT*	

NXelectrostatic_kicker/NXlog Members Declarations in the *NXelectrostatic_kicker/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_VOLTAGE*	

NXfresnel_zone_plate

version	category	extends	groups cited
1.0	base	*NXobject*	*NXgeometry*

description for a fresnel zone plate

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXfresnel_zone_plate.nxdl.xml

svnid: Id

Structure of NXfresnel_zone_plate

```
NXfresnel_zone_plate (base class, version 1.0)
  (base class definition, NXentry or NXsubentry not found)
  central_stop_diameter:NX_FLOAT
  central_stop_material:NX_CHAR
  central_stop_thickness:NX_FLOAT
  fabrication:NX_CHAR
  focus_parameters:NX_FLOAT
  mask_material:NX_CHAR
  mask_thickness:NX_FLOAT
  outer_diameter:NX_FLOAT
  outermost_zone_width:NX_FLOAT
  support_membrane_material:NX_CHAR
  support_membrane_thickness:NX_FLOAT
  zone_height:NX_FLOAT
  zone_material:NX_CHAR
```

```
16    zone_support_material:NX_CHAR
17    NXgeometry
```

NXfresnel_zone_plate Members Declarations in the *NXfresnel_zone_plate* group.

Name and Attributes	Type	Units	Description (and Occurrences)
focus_parameters	*NX_FLOAT*		list of polynomial coefficients describing the focal length of the zone plate, in increasing order.
outer_diameter	*NX_FLOAT*	*NX_LENGTH*	
outermost_zone_width	*NX_FLOAT*	*NX_LENGTH*	
central_stop_diameter	*NX_FLOAT*	*NX_LENGTH*	
fabrication	(*NX_CHAR*)		how zone plate was manufactured Any of these value(s): • `etched`: • `plated`: • `zone doubled`:
zone_height	*NX_FLOAT*	*NX_LENGTH*	
zone_material	(*NX_CHAR*)		Material of the zones themselves
zone_support_material	(*NX_CHAR*)		Material present between the zones. This is usually only present for the "zone doubled" fabrication process
central_stop_material	(*NX_CHAR*)		
central_stop_thickness	*NX_FLOAT*	*NX_LENGTH*	
mask_thickness	*NX_FLOAT*	*NX_LENGTH*	
mask_material	(*NX_CHAR*)		If no mask is present, set mask_thickness to 0 and omit the mask_material field
support_membrane_material	(*NX_CHAR*)		
support_membrane_thickness	*NX_FLOAT*	*NX_LENGTH*	
(geometry)	*NXgeometry*		"Engineering" position of the fresnel zone plate

NXmagnetic_kicker

version	category	extends	groups cited
1.0	contributed	*NXobject*	*NXlog*

definition for a magnetic kicker.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXmagnetic_kicker.nxdl.xml

svnid: Id: NXmagnetic_kicker.nxdl.xml 1162 2012-09-27 03:57:58Z Pete Jemian

Structure of NXmagnetic_kicker

```
1  NXmagnetic_kicker (contributed definition, version 1.0)
2    (base class definition, NXentry or NXsubentry not found)
3    beamline_distance:NX_FLOAT
4    description:NX_CHAR
5    set_current:NX_FLOAT
6    set_voltage:NX_FLOAT
7    timing:NX_FLOAT
8      @description
```

```
  read_current:NXlog
    value:NX_CHAR
  read_voltage:NXlog
    value:NX_CHAR
```

NXmagnetic_kicker Members Declarations in the *NXmagnetic_kicker* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	*NX_CHAR*		extended description of the kicker.
beamline_distance	*NX_FLOAT*	*NX_LENGTH*	define position of beamline element relative to production target
timing	*NX_FLOAT*	*NX_TIME*	kicker timing as defined by `description` attribute
@description	*NX_CHAR*		
set_current	*NX_FLOAT*	*NX_CURRENT*	current set on supply.
set_voltage	*NX_FLOAT*	*NX_VOLTAGE*	voltage set on supply.
read_current	*NXlog*		current read from supply. See table *NXmagnetic_kicker/NXlog Members*.
read_voltage	*NXlog*		voltage read from supply. See table *NXmagnetic_kicker/NXlog Members*.

NXmagnetic_kicker/NXlog Members Declarations in the *NXmagnetic_kicker/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_CURRENT*	

NXmagnetic_kicker/NXlog Members Declarations in the *NXmagnetic_kicker/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_VOLTAGE*	

NXquadrupole_magnet

version	category	extends	groups cited
1.0	contributed	*NXobject*	*NXlog*

definition for a quadrupole magnet.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXquadrupole_magnet.nxdl.xml

svnid: Id: NXquadrupole_magnet.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXquadrupole_magnet

```
NXquadrupole_magnet (contributed definition, version 1.0)
  (base class definition, NXentry or NXsubentry not found)
  beamline_distance:NX_FLOAT
  description:NX_CHAR
  set_current:NX_FLOAT
  read_current:NXlog
    value:NX_CHAR
  read_voltage:NXlog
    value:NX_CHAR
```

NXquadrupole_magnet Members Declarations in the *NXquadrupole_magnet* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	*NX_CHAR*		extended description of the magnet.
beamline_distance	*NX_FLOAT*	*NX_LENGTH*	define position of beamline element relative to production target
set_current	*NX_FLOAT*	*NX_CURRENT*	current set on supply.
read_current	*NXlog*		current read from supply. See table *NXquadrupole_magnet/NXlog Members*.
read_voltage	*NXlog*		voltage read from supply. See table *NXquadrupole_magnet/NXlog Members*.

NXquadrupole_magnet/NXlog Members Declarations in the *NXquadrupole_magnet/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_CURRENT*	

NXquadrupole_magnet/NXlog Members Declarations in the *NXquadrupole_magnet/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_VOLTAGE*	

NXseparator

version	category	extends	groups cited
1.0	contributed	*NXobject*	*NXlog*

definition for an electrostatic separator.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXseparator.nxdl.xml

svnid: Id: NXseparator.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXseparator

```
NXseparator (contributed definition, version 1.0)
  (base class definition, NXentry or NXsubentry not found)
  beamline_distance:NX_FLOAT
  description:NX_CHAR
  set_Bfield_current:NX_FLOAT
  set_Efield_voltage:NX_FLOAT
  read_Bfield_current:NXlog
    value:NX_CHAR
  read_Bfield_voltage:NXlog
    value:NX_CHAR
  read_Efield_current:NXlog
    value:NX_CHAR
  read_Efield_voltage:NXlog
    value:NX_CHAR
```

NXseparator Members Declarations in the *NXseparator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	*NX_CHAR*		extended description of the separator.
beamline_distance	*NX_FLOAT*	*NX_LENGTH*	define position of beamline element relative to production target
set_Bfield_current	*NX_FLOAT*	*NX_CURRENT*	current set on magnet supply.
set_Efield_voltage	*NX_FLOAT*	*NX_VOLTAGE*	current set on HT supply.
read_Bfield_current	*NXlog*		current read from magnet supply. See table *NXseparator/NXlog Members*.
read_Bfield_voltage	*NXlog*		voltage read from magnet supply. See table *NXseparator/NXlog Members*.
read_Efield_current	*NXlog*		current read from HT supply. See table *NXseparator/NXlog Members*.
read_Efield_voltage	*NXlog*		voltage read from HT supply. See table *NXseparator/NXlog Members*.

NXseparator/NXlog Members Declarations in the *NXseparator/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_CURRENT*	

NXseparator/NXlog Members Declarations in the *NXseparator/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_VOLTAGE*	

NXseparator/NXlog Members Declarations in the *NXseparator/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_CURRENT*	

NXseparator/NXlog Members Declarations in the *NXseparator/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_VOLTAGE*	

NXsnsevent

version	category	extends	groups cited
1.0	contributed	*NXobject*	*NXaperture*, *NXattenuator*, *NXcollection*, *NXcrystal*, *NXdata*, *NXdetector*, *NXdisk_chopper*, *NXentry*, *NXevent_data*, *NXgeometry*, *NXinstrument*, *NXlog*, *NXmoderator*, *NXmonitor*, *NXnote*, *NXorientation*, *NXpolarizer*, *NXpositioner*, *NXsample*, *NXshape*, *NXsource*, *NXtranslation*, *NXuser*

This is a definition for event data from Spallation Neutron Source (SNS) at ORNL.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXsnsevent.nxdl.xml

svnid: Id: NXsnsevent.nxdl.xml 1190 2013-01-10 20:17:07Z Pete Jemian

Structure of NXsnsevent

```
NXsnsevent (contributed definition, version 1.0)
  (overlays NXentry)
  NXentry
    collection_identifier:NX_CHAR
    collection_title:NX_CHAR
    definition:NX_CHAR
    duration:NX_FLOAT
    end_time:NX_DATE_TIME
    entry_identifier:NX_CHAR
    experiment_identifier:NX_CHAR
    notes:NX_CHAR
    proton_charge:NX_FLOAT
    raw_frames:NX_INT
    run_number:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    total_counts:NX_UINT
    total_uncounted_counts:NX_UINT
    DASlogs:NXcollection
      NXlog
        average_value:NX_FLOAT
        average_value_error:NX_FLOAT
        description:NX_CHAR
        duration:NX_FLOAT
        maximum_value:NX_FLOAT
        minimum_value:NX_FLOAT
        time:NX_FLOAT[nvalue]
        value:NX_FLOAT[nvalue]
      NXpositioner
        average_value:NX_FLOAT
        average_value_error:NX_FLOAT
        description:NX_CHAR
        duration:NX_FLOAT
        maximum_value:NX_FLOAT
        minimum_value:NX_FLOAT
        time:NX_FLOAT[numvalue]
        value:NX_FLOAT[numvalue]
    NXdata
      data_x_y --> /NXentry/NXinstrument/NXdetector/data_x_y
      x_pixel_offset --> /NXentry/NXinstrument/NXdetector/x_pixel_offset
      y_pixel_offset --> /NXentry/NXinstrument/NXdetector/y_pixel_offset
    NXevent_data
      event_index --> /NXentry/NXinstrument/NXdetector/event_index
      event_pixel_id --> /NXentry/NXinstrument/NXdetector/event_pixel_id
      event_time_of_flight --> /NXentry/NXinstrument/NXdetector/event_time_of_flight
      pulse_time --> /NXentry/NXinstrument/NXdetector/pulse_time
    instrument:NXinstrument
      SNSdetector_calibration_id:NX_CHAR
      SNSgeometry_file_name:NX_CHAR
      SNStranslation_service:NX_CHAR
      beamline:NX_CHAR
      name:NX_CHAR
      NXaperture
        x_pixel_offset:NX_FLOAT
        origin:NXgeometry
          orientation:NXorientation
```

```
            value:NX_FLOAT[6]
          shape:NXshape
            description:NX_CHAR
            shape:NX_CHAR
            size:NX_FLOAT[3]
          translation:NXtranslation
            distance:NX_FLOAT[3]
      NXattenuator
        distance:NX_FLOAT
      NXcrystal
        type:NX_CHAR
        wavelength:NX_FLOAT
        origin:NXgeometry
          description:NX_CHAR
          orientation:NXorientation
            value:NX_FLOAT[6]
          shape:NXshape
            description:NX_CHAR
            shape:NX_CHAR
            size:NX_FLOAT
          translation:NXtranslation
            distance:NX_FLOAT[3]
      NXdetector
        azimuthal_angle:NX_FLOAT[numx,numy]
        data_x_y:NX_UINT[numx,numy]
        distance:NX_FLOAT[numx,numy]
        event_index:NX_UINT[numpulses]
        event_pixel_id:NX_UINT[numevents]
        event_time_of_flight:NX_FLOAT[numevents]
        pixel_id:NX_UINT[numx,numy]
        polar_angle:NX_FLOAT[numx,numy]
        pulse_time:NX_FLOAT[numpulses]
        total_counts:NX_UINT
        x_pixel_offset:NX_FLOAT[numx]
        y_pixel_offset:NX_FLOAT[numy]
        origin:NXgeometry
          orientation:NXorientation
            value:NX_FLOAT[6]
          shape:NXshape
            description:NX_CHAR
            shape:NX_CHAR
            size:NX_FLOAT[3]
          translation:NXtranslation
            distance:NX_FLOAT[3]
      NXdisk_chopper
        distance:NX_FLOAT
      moderator:NXmoderator
        coupling_material:NX_CHAR
        distance:NX_FLOAT
        temperature:NX_FLOAT
        type:NX_CHAR
      NXpolarizer
      SNS:NXsource
        frequency:NX_FLOAT
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    NXmonitor
```

```
      data:NX_UINT[numtimechannels]
      distance:NX_FLOAT
      mode:NX_CHAR
      time_of_flight:NX_FLOAT[numtimechannels + 1]
    SNSHistoTool:NXnote
      SNSbanking_file_name:NX_CHAR
      SNSmapping_file_name:NX_CHAR
      author:NX_CHAR
      command1:NX_CHAR
      date:NX_CHAR
      description:NX_CHAR
      version:NX_CHAR
    sample:NXsample
      changer_position:NX_CHAR
      holder:NX_CHAR
      identifier:NX_CHAR
      name:NX_CHAR
      nature:NX_CHAR
    NXuser
      facility_user_id:NX_CHAR
      name:NX_CHAR
      role:NX_CHAR
```

NXsnsevent Members Declarations in the *NXsnsevent* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXsnsevent/NXentry Members*.

NXsnsevent/NXentry Members Declarations in the *NXsnsevent/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
collection_identifier	(*NX_CHAR*)		
collection_title	(*NX_CHAR*)		
definition	(*NX_CHAR*)		Official NXDL schema after this file goes to applications. This value: `NXsnsevent`
duration	*NX_FLOAT*	*NX_TIME*	
end_time	*NX_DATE_TIME*		
entry_identifier	(*NX_CHAR*)		
experiment_identifier	(*NX_CHAR*)		
notes	(*NX_CHAR*)		
proton_charge	*NX_FLOAT*	*NX_CHARGE*	
raw_frames	*NX_INT*		
run_number	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
title	(*NX_CHAR*)		
total_counts	*NX_UINT*	*NX_UNITLESS*	
total_uncounted_counts	*NX_UINT*	*NX_UNITLESS*	
DASlogs	*NXcollection*		Details of all logs, both from cvinfo file and from HistoTool (frequency and proton_charge). See table *NXsnsevent/NXentry/NXcollection Members*.
SNSHistoTool	*NXnote*		See table *NXsnsevent/NXentry/NXnote Members*.
(data)	*NXdata*		See table *NXsnsevent/NXentry/NXdata Members*.
(event_data)	*NXevent_data*		See table *NXsnsevent/NXentry/NXevent_data Members*.
instrument	*NXinstrument*		See table *NXsnsevent/NXentry/NXinstrument Members*.
(monitor)	*NXmonitor*		See table *NXsnsevent/NXentry/NXmonitor Members*.
sample	*NXsample*		See table *NXsnsevent/NXentry/NXsample Members*.
(user)	*NXuser*		See table *NXsnsevent/NXentry/NXuser Members*.

NXsnsevent/NXentry/NXcollection Members Declarations in the *NXsnsevent/NXentry/NXcollection* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(log)	*NXlog*		See table *NXsnsevent/NXentry/NXcollection/NXlog Members*.
(positioner)	*NXpositioner*		Motor logs from cvinfo file. See table *NXsnsevent/NXentry/NXcollection/NXpositioner Members*.

NXsnsevent/NXentry/NXnote Members Declarations in the *NXsnsevent/NXentry/NXnote* group.

Name and Attributes	Type	Units	Description (and Occurrences)
SNSbanking_file_name	(*NX_CHAR*)		
SNSmapping_file_name	(*NX_CHAR*)		
author	(*NX_CHAR*)		
command1	(*NX_CHAR*)		Command string for event2nxl.
date	(*NX_CHAR*)		
description	(*NX_CHAR*)		
version	(*NX_CHAR*)		

NXsnsevent/NXentry/NXdata Members Declarations in the *NXsnsevent/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data_x_y	*link*		target = `/NXentry/NXinstrument/NXdetecto`
x_pixel_offset	*link*		target = `/NXentry/NXinstrument/NXdetecto`
y_pixel_offset	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXsnsevent/NXentry/NXevent_data Members Declarations in the *NXsnsevent/NXentry/NXevent_data* group.

Name and Attributes	Type	Units	Description (and Occurrences)
event_index	*link*		target = `/NXentry/NXinstrument/NXdetecto`
event_pixel_id	*link*		target = `/NXentry/NXinstrument/NXdetecto`
event_time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`
pulse_time	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXsnsevent/NXentry/NXinstrument Members Declarations in the *NXsnsevent/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
SNSdetector_calibration_id	(*NX_CHAR*)		Detector calibration id from DAS.
SNSgeometry_file_name	(*NX_CHAR*)		
SNStranslation_service	(*NX_CHAR*)		
beamline	(*NX_CHAR*)		
name	(*NX_CHAR*)		
SNS	*NXsource*		See table *NXsnsevent/NXentry/NXinstrument/NXsource Members*.
(detector)	*NXdetector*		See table *NXsnsevent/NXentry/NXinstrument/NXdetector Members*.
(disk_chopper)	*NXdisk_chopper*		See table *NXsnsevent/NXentry/NXinstrument/NXdisk_chopper Members*.
moderator	*NXmoderator*		See table *NXsnsevent/NXentry/NXinstrument/NXmoderator Members*.
(aperture)	*NXaperture*		See table *NXsnsevent/NXentry/NXinstrument/NXaperture Members*.
(attenuator)	*NXattenuator*		See table *NXsnsevent/NXentry/NXinstrument/NXattenuator Members*.
(polarizer)	*NXpolarizer*		
(crystal)	*NXcrystal*		See table *NXsnsevent/NXentry/NXinstrument/NXcrystal Members*.

NXsnsevent/NXentry/NXmonitor Members Declarations in the *NXsnsevent/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_UINT*		expect `signal=1` `axes="time_of_flight"` Dimensions: (rank=1) • 1: `numtimechannels`
distance	*NX_FLOAT*	*NX_LENGTH*	
mode	(*NX_CHAR*)		
time_of_flight	*NX_FLOAT*	*NX_TIME*	Dimensions: (rank=1) • 1: `numtimechannels + 1`

NXsnsevent/NXentry/NXsample Members Declarations in the *NXsnsevent/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
changer_position	(*NX_CHAR*)		
holder	(*NX_CHAR*)		
identifier	(*NX_CHAR*)		
name	(*NX_CHAR*)		Descriptive name of sample
nature	(*NX_CHAR*)		

NXsnsevent/NXentry/NXuser Members Declarations in the *NXsnsevent/NXentry/NXuser* group.

Name and Attributes	Type	Units	Description (and Occurrences)
facility_user_id	(*NX_CHAR*)		
name	(*NX_CHAR*)		
role	(*NX_CHAR*)		

NXsnsevent/NXentry/NXcollection/NXlog Members Declarations in the *NXsnsevent/NXentry/NXcollection/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
average_value	*NX_FLOAT*		
average_value_error	*NX_FLOAT*		
description	(*NX_CHAR*)		
duration	*NX_FLOAT*		
maximum_value	*NX_FLOAT*		
minimum_value	*NX_FLOAT*		
time	*NX_FLOAT*		Dimensions: (rank=1) • 1: `nvalue`
value	*NX_FLOAT*		Dimensions: (rank=1) • 1: `nvalue`

NXsnsevent/NXentry/NXcollection/NXpositioner Members Declarations in the *NXsnsevent/NXentry/NXcollection/NXpositioner* group.

Name and Attributes	Type	Units	Description (and Occurrences)
average_value	*NX_FLOAT*		
average_value_error	*NX_FLOAT*		
description	(*NX_CHAR*)		
duration	*NX_FLOAT*		
maximum_value	*NX_FLOAT*		
minimum_value	*NX_FLOAT*		
time	*NX_FLOAT*		Dimensions: (rank=1) • 1: `numvalue`
value	*NX_FLOAT*		Dimensions: (rank=1) • 1: `numvalue`

NXsnsevent/NXentry/NXinstrument/NXsource Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
frequency	*NX_FLOAT*	*NX_FREQUENCY*	
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		
type	(*NX_CHAR*)		

NXsnsevent/NXentry/NXinstrument/NXdetector Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=2) • 1: `numx` • 2: `numy`
data_x_y	*NX_UINT*		expect `signal=2 axes="x_pixel_offset,y_pixel_offset"` Dimensions: (rank=2) • 1: `numx` • 2: `numy`
distance	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=2) • 1: `numx` • 2: `numy`
event_index	*NX_UINT*		Dimensions: (rank=1) • 1: `numpulses`
event_pixel_id	*NX_UINT*		Dimensions: (rank=1) • 1: `numevents`
event_time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `numevents`
pixel_id	*NX_UINT*		Dimensions: (rank=2) • 1: `numx` • 2: `numy`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=2) • 1: `numx` • 2: `numy`
pulse_time	*NX_FLOAT*	*NX_TIME*	Dimensions: (rank=1) • 1: `numpulses`
total_counts	*NX_UINT*		
x_pixel_offset	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `numx`
y_pixel_offset	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `numy`
origin	*NXgeometry*		See table *NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry Members*.

NXsnsevent/NXentry/NXinstrument/NXdisk_chopper Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXdisk_chopper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	

NXsnsevent/NXentry/NXinstrument/NXmoderator Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXmoderator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
coupling_material	(*NX_CHAR*)		
distance	*NX_FLOAT*	*NX_LENGTH*	
temperature	*NX_FLOAT*	*NX_TEMPERATURE*	
type	(*NX_CHAR*)		

NXsnsevent/NXentry/NXinstrument/NXaperture Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXaperture* group.

Name and Attributes	Type	Units	Description (and Occurrences)
x_pixel_offset	*NX_FLOAT*	*NX_LENGTH*	
origin	*NXgeometry*		See table *NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry Members*.

NXsnsevent/NXentry/NXinstrument/NXattenuator Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXattenuator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	

NXsnsevent/NXentry/NXinstrument/NXcrystal Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXcrystal* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	
origin	*NXgeometry*		See table *NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry Members*.

NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
orientation	*NXorientation*		See table *NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/N Members*.
shape	*NXshape*		See table *NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/N Members*.
translation	*NXtranslation*		See table *NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/N Members*.

NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
orientation	*NXorientation*		See table *NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/N Members*.
shape	*NXshape*		See table *NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/N Members*.
translation	*NXtranslation*		See table *NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/N Members*.

NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		
orientation	*NXorientation*		See table *NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NX Members*.
shape	*NXshape*		See table *NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NX Members*.
translation	*NXtranslation*		See table *NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NX Members*.

NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/NXorientation Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/NXorientation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	*NX_FLOAT*		Six out of nine rotation parameters. Dimensions: (rank=1) • 1: `6`

NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/NXshape Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		
shape	(*NX_CHAR*)		
size	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `3`

NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/NXtranslation Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXdetector/NXgeometry/NXtranslation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `3`

NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/NXorientation Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/NXorientation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	*NX_FLOAT*		Six out of nine rotation parameters. Dimensions: (rank=1) • 1: 6

NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/NXshape Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		
shape	(*NX_CHAR*)		
size	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: 3

NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/NXtranslation Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXaperture/NXgeometry/NXtranslation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: 3

NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NXorientation Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NXorientation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	*NX_FLOAT*		Six out of nine rotation parameters. Dimensions: (rank=1) • 1: 6

NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NXshape Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		
shape	(*NX_CHAR*)		
size	*NX_FLOAT*	*NX_LENGTH*	

NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NXtranslation Members Declarations in the *NXsnsevent/NXentry/NXinstrument/NXcrystal/NXgeometry/NXtranslation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: 3

	version	category	extends	groups cited
NXsnshisto	1.0	contributed	*NXobject*	*NXaperture*, *NXattenuator*, *NXcollection*, *NXcrystal*, *NXdata*, *NXdetector*, *NXdisk_chopper*, *NXentry*, *NXfermi_chopper*, *NXgeometry*, *NXinstrument*, *NXlog*, *NXmoderator*, *NXmonitor*, *NXnote*, *NXorientation*, *NXpolarizer*, *NXpositioner*, *NXsample*, *NXshape*, *NXsource*, *NXtranslation*, *NXuser*

This is a definition for histogram data from Spallation Neutron Source (SNS) at ORNL.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXsnshisto.nxdl.xml

svnid: Id: NXsnshisto.nxdl.xml 1045 2012-02-13 00:10:56Z Freddie Akeroyd

Structure of NXsnshisto

```
NXsnshisto (contributed definition, version 1.0)
  (overlays NXentry)
  NXentry
    collection_identifier:NX_CHAR
    collection_title:NX_CHAR
    definition:NX_CHAR
    duration:NX_FLOAT
    end_time:NX_DATE_TIME
    entry_identifier:NX_CHAR
    experiment_identifier:NX_CHAR
    notes:NX_CHAR
    proton_charge:NX_FLOAT
    raw_frames:NX_INT
    run_number:NX_CHAR
    start_time:NX_DATE_TIME
    title:NX_CHAR
    total_counts:NX_UINT
    total_uncounted_counts:NX_UINT
    DASlogs:NXcollection
      NXlog
        average_value:NX_FLOAT
        average_value_error:NX_FLOAT
        description:NX_CHAR
        duration:NX_FLOAT
        maximum_value:NX_FLOAT
        minimum_value:NX_FLOAT
        time:NX_FLOAT[nvalue]
        value:NX_FLOAT[nvalue]
      NXpositioner
        average_value:NX_FLOAT
        average_value_error:NX_FLOAT
        description:NX_CHAR
        duration:NX_FLOAT
        maximum_value:NX_FLOAT
        minimum_value:NX_FLOAT
        time:NX_FLOAT[numvalue]
        value:NX_FLOAT[numvalue]
    NXdata
      data --> /NXentry/NXinstrument/NXdetector/data
      data_x_time_of_flight --> /NXentry/NXinstrument/NXdetector/data_x_time_of_flight
      data_x_y --> /NXentry/NXinstrument/NXdetector/data_x_y
```

```
      data_y_time_of_flight --> /NXentry/NXinstrument/NXdetector/data_y_time_of_flight
      pixel_id --> /NXentry/NXinstrument/NXdetector/pixel_id
      time_of_flight --> /NXentry/NXinstrument/NXdetector/time_of_flight
      total_counts --> /NXentry/NXinstrument/NXdetector/total_counts
      x_pixel_offset --> /NXentry/NXinstrument/NXdetector/x_pixel_offset
      y_pixel_offset --> /NXentry/NXinstrument/NXdetector/y_pixel_offset
    instrument:NXinstrument
      SNSdetector_calibration_id:NX_CHAR
      SNSgeometry_file_name:NX_CHAR
      SNStranslation_service:NX_CHAR
      beamline:NX_CHAR
      name:NX_CHAR
      NXaperture
        x_pixel_offset:NX_FLOAT
        origin:NXgeometry
          orientation:NXorientation
            value:NX_FLOAT[6]
          shape:NXshape
            description:NX_CHAR
            shape:NX_CHAR
            size:NX_FLOAT[3]
          translation:NXtranslation
            distance:NX_FLOAT[3]
      NXattenuator
        distance:NX_FLOAT
      NXcrystal
        type:NX_CHAR
        wavelength:NX_FLOAT
        origin:NXgeometry
          description:NX_CHAR
          orientation:NXorientation
            value:NX_FLOAT[6]
          shape:NXshape
            description:NX_CHAR
            shape:NX_CHAR
            size:NX_FLOAT
          translation:NXtranslation
            distance:NX_FLOAT[3]
      NXdetector
        azimuthal_angle:NX_FLOAT[numx,numy]
        data:NX_UINT[numx,numy,numtof]
        data_x_time_of_flight:NX_UINT[numx,numtof]
        data_x_y:NX_UINT[numx,numy]
        data_y_time_of_flight:NX_UINT[numy,numtof]
        distance:NX_FLOAT[numx,numy]
        pixel_id:NX_UINT[numx,numy]
        polar_angle:NX_FLOAT[numx,numy]
        time_of_flight:NX_FLOAT[numtof + 1]
        total_counts:NX_UINT
        x_pixel_offset:NX_FLOAT[numx]
        y_pixel_offset:NX_FLOAT[numy]
        origin:NXgeometry
          orientation:NXorientation
            value:NX_FLOAT[6]
          shape:NXshape
            description:NX_CHAR
            shape:NX_CHAR
            size:NX_FLOAT[3]
```

```
          translation:NXtranslation
            distance:NX_FLOAT[3]
      NXdisk_chopper
        distance:NX_FLOAT
      NXfermi_chopper
        distance:NX_FLOAT
      moderator:NXmoderator
        coupling_material:NX_CHAR
        distance:NX_FLOAT
        temperature:NX_FLOAT
        type:NX_CHAR
      NXpolarizer
      SNS:NXsource
        frequency:NX_FLOAT
        name:NX_CHAR
        probe:NX_CHAR
        type:NX_CHAR
    NXmonitor
      data:NX_UINT[numtimechannels]
      distance:NX_FLOAT
      mode:NX_CHAR
      time_of_flight:NX_FLOAT[numtimechannels + 1]
    SNSHistoTool:NXnote
      SNSbanking_file_name:NX_CHAR
      SNSmapping_file_name:NX_CHAR
      author:NX_CHAR
      command1:NX_CHAR
      date:NX_CHAR
      description:NX_CHAR
      version:NX_CHAR
    sample:NXsample
      changer_position:NX_CHAR
      holder:NX_CHAR
      identifier:NX_CHAR
      name:NX_CHAR
      nature:NX_CHAR
    NXuser
      facility_user_id:NX_CHAR
      name:NX_CHAR
      role:NX_CHAR
```

NXsnshisto Members Declarations in the *NXsnshisto* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(entry)	*NXentry*		See table *NXsnshisto/NXentry Members*.

NXsnshisto/NXentry Members Declarations in the *NXsnshisto/NXentry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
collection_identifier	(*NX_CHAR*)		
collection_title	(*NX_CHAR*)		
definition	(*NX_CHAR*)		Official NXDL schema after this file goes to applications. This value: `NXsnshisto`
duration	*NX_FLOAT*	*NX_TIME*	
end_time	*NX_DATE_TIME*		
entry_identifier	(*NX_CHAR*)		
experiment_identifier	(*NX_CHAR*)		
notes	(*NX_CHAR*)		
proton_charge	*NX_FLOAT*	*NX_CHARGE*	
raw_frames	*NX_INT*		
run_number	(*NX_CHAR*)		
start_time	*NX_DATE_TIME*		
title	(*NX_CHAR*)		
total_counts	*NX_UINT*	*NX_UNITLESS*	
total_uncounted_counts	*NX_UINT*	*NX_UNITLESS*	
DASlogs	*NXcollection*		Details of all logs, both from cvinfo file and from HistoTool (frequency and proton_charge). See table *NXsnshisto/NXentry/NXcollection Members*.
SNSHistoTool	*NXnote*		See table *NXsnshisto/NXentry/NXnote Members*.
(data)	*NXdata*		See table *NXsnshisto/NXentry/NXdata Members*.
instrument	*NXinstrument*		See table *NXsnshisto/NXentry/NXinstrument Members*.
(monitor)	*NXmonitor*		See table *NXsnshisto/NXentry/NXmonitor Members*.
sample	*NXsample*		See table *NXsnshisto/NXentry/NXsample Members*.
(user)	*NXuser*		See table *NXsnshisto/NXentry/NXuser Members*.

NXsnshisto/NXentry/NXcollection Members Declarations in the *NXsnshisto/NXentry/NXcollection* group.

Name and Attributes	Type	Units	Description (and Occurrences)
(log)	*NXlog*		See table *NXsnshisto/NXentry/NXcollection/NXlog Members*.
(positioner)	*NXpositioner*		Motor logs from cvinfo file. See table *NXsnshisto/NXentry/NXcollection/NXpositio Members*.

NXsnshisto/NXentry/NXnote Members Declarations in the *NXsnshisto/NXentry/NXnote* group.

Name and Attributes	Type	Units	Description (and Occurrences)
SNSbanking_file_name	(*NX_CHAR*)		
SNSmapping_file_name	(*NX_CHAR*)		
author	(*NX_CHAR*)		
command1	(*NX_CHAR*)		Command string for event2histo_nxl.
date	(*NX_CHAR*)		
description	(*NX_CHAR*)		
version	(*NX_CHAR*)		

NXsnshisto/NXentry/NXdata Members Declarations in the *NXsnshisto/NXentry/NXdata* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*link*		target = `/NXentry/NXinstrument/NXdetecto`
data_x_time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`
data_x_y	*link*		target = `/NXentry/NXinstrument/NXdetecto`
data_y_time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`
pixel_id	*link*		target = `/NXentry/NXinstrument/NXdetecto`
time_of_flight	*link*		target = `/NXentry/NXinstrument/NXdetecto`
total_counts	*link*		target = `/NXentry/NXinstrument/NXdetecto`
x_pixel_offset	*link*		target = `/NXentry/NXinstrument/NXdetecto`
y_pixel_offset	*link*		target = `/NXentry/NXinstrument/NXdetecto`

NXsnshisto/NXentry/NXinstrument Members Declarations in the *NXsnshisto/NXentry/NXinstrument* group.

Name and Attributes	Type	Units	Description (and Occurrences)
SNSdetector_calibration_id	(*NX_CHAR*)		Detector calibration id from DAS.
SNSgeometry_file_name	(*NX_CHAR*)		
SNStranslation_service	(*NX_CHAR*)		
beamline	(*NX_CHAR*)		
name	(*NX_CHAR*)		
SNS	*NXsource*		See table *NXsnshisto/NXentry/NXinstrument/NXsource Members*.
(detector)	*NXdetector*		See table *NXsnshisto/NXentry/NXinstrument/NXdetect Members*.
(disk_chopper)	*NXdisk_chopper*		Original specification called for NXchopper, which is not a valid NeXus base class. Select either NXdisk_chopper or NXfermi_chopper, as appropriate. See table *NXsnshisto/NXentry/NXinstrument/NXdisk_c Members*.
(fermi_chopper)	*NXfermi_chopper*		Original specification called for NXchopper, which is not a valid NeXus base class. Select either NXdisk_chopper or NXfermi_chopper, as appropriate. See table *NXsnshisto/NXentry/NXinstrument/NXfermi_ Members*.
moderator	*NXmoderator*		See table *NXsnshisto/NXentry/NXinstrument/NXmoder Members*.
(aperture)	*NXaperture*		See table *NXsnshisto/NXentry/NXinstrument/NXapertu Members*.
(attenuator)	*NXattenuator*		See table *NXsnshisto/NXentry/NXinstrument/NXattenu Members*.
(polarizer)	*NXpolarizer*		
(crystal)	*NXcrystal*		See table *NXsnshisto/NXentry/NXinstrument/NXcrysta Members*.

NXsnshisto/NXentry/NXmonitor Members Declarations in the *NXsnshisto/NXentry/NXmonitor* group.

Name and Attributes	Type	Units	Description (and Occurrences)
data	*NX_UINT*		Dimensions: (rank=1) • 1: `numtimechannels`
distance	*NX_FLOAT*	*NX_LENGTH*	
mode	(*NX_CHAR*)		
time_of_flight	*NX_FLOAT*	*NX_TIME*	Dimensions: (rank=1) • 1: `numtimechannels + 1`

NXsnshisto/NXentry/NXsample Members Declarations in the *NXsnshisto/NXentry/NXsample* group.

Name and Attributes	Type	Units	Description (and Occurrences)
changer_position	(*NX_CHAR*)		
holder	(*NX_CHAR*)		
identifier	(*NX_CHAR*)		
name	(*NX_CHAR*)		Descriptive name of sample
nature	(*NX_CHAR*)		

NXsnshisto/NXentry/NXuser Members Declarations in the *NXsnshisto/NXentry/NXuser* group.

Name and Attributes	Type	Units	Description (and Occurrences)
facility_user_id	(*NX_CHAR*)		
name	(*NX_CHAR*)		
role	(*NX_CHAR*)		

NXsnshisto/NXentry/NXcollection/NXlog Members Declarations in the *NXsnshisto/NXentry/NXcollection/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
average_value	*NX_FLOAT*		
average_value_error	*NX_FLOAT*		
description	(*NX_CHAR*)		
duration	*NX_FLOAT*		
maximum_value	*NX_FLOAT*		
minimum_value	*NX_FLOAT*		
time	*NX_FLOAT*		Dimensions: (rank=1) • 1: `nvalue`
value	*NX_FLOAT*		Dimensions: (rank=1) • 1: `nvalue`

NXsnshisto/NXentry/NXcollection/NXpositioner Members Declarations in the *NXsnshisto/NXentry/NXcollection/NXpositioner* group.

Name and Attributes	Type	Units	Description (and Occurrences)
average_value	*NX_FLOAT*		
average_value_error	*NX_FLOAT*		
description	(*NX_CHAR*)		
duration	*NX_FLOAT*		
maximum_value	*NX_FLOAT*		
minimum_value	*NX_FLOAT*		
time	*NX_FLOAT*		Dimensions: (rank=1) • 1: `numvalue`
value	*NX_FLOAT*		Dimensions: (rank=1) • 1: `numvalue`

NXsnshisto/NXentry/NXinstrument/NXsource Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXsource* group.

Name and Attributes	Type	Units	Description (and Occurrences)
frequency	*NX_FLOAT*	*NX_FREQUENCY*	
name	(*NX_CHAR*)		
probe	(*NX_CHAR*)		
type	(*NX_CHAR*)		

NXsnshisto/NXentry/NXinstrument/NXdetector Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXdetector* group.

Name and Attributes	Type	Units	Description (and Occurrences)
azimuthal_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=2) • 1: `numx` • 2: `numy`
data	*NX_UINT*		Dimensions: (rank=3) • 1: `numx` • 2: `numy` • 3: `numtof`
data_x_time_of_flight	*NX_UINT*		Dimensions: (rank=2) • 1: `numx` • 2: `numtof`
data_x_y	*NX_UINT*		Dimensions: (rank=2) • 1: `numx` • 2: `numy`
data_y_time_of_flight	*NX_UINT*		Dimensions: (rank=2) • 1: `numy` • 2: `numtof`
distance	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=2) • 1: `numx` • 2: `numy`
pixel_id	*NX_UINT*		Dimensions: (rank=2) • 1: `numx` • 2: `numy`
polar_angle	*NX_FLOAT*	*NX_ANGLE*	Dimensions: (rank=2) • 1: `numx` • 2: `numy`
time_of_flight	*NX_FLOAT*	*NX_TIME_OF_FLIGHT*	Dimensions: (rank=1) • 1: `numtof + 1`
total_counts	*NX_UINT*		
x_pixel_offset	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `numx`
y_pixel_offset	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: `numy`
origin	*NXgeometry*		See table *NXsnshisto/NXentry/NXinstrument/NXdetect Members*.

NXsnshisto/NXentry/NXinstrument/NXdisk_chopper Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXdisk_chopper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	

NXsnshisto/NXentry/NXinstrument/NXfermi_chopper Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXfermi_chopper* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	

NXsnshisto/NXentry/NXinstrument/NXmoderator Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXmoderator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
coupling_material	(*NX_CHAR*)		
distance	*NX_FLOAT*	*NX_LENGTH*	
temperature	*NX_FLOAT*	*NX_TEMPERATURE*	
type	(*NX_CHAR*)		

NXsnshisto/NXentry/NXinstrument/NXaperture Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXaperture* group.

Name and Attributes	Type	Units	Description (and Occurrences)
x_pixel_offset	*NX_FLOAT*	*NX_LENGTH*	
origin	*NXgeometry*		See table *NXsnshisto/NXentry/NXinstrument/NXapertu Members*.

NXsnshisto/NXentry/NXinstrument/NXattenuator Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXattenuator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	

NXsnshisto/NXentry/NXinstrument/NXcrystal Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXcrystal* group.

Name and Attributes	Type	Units	Description (and Occurrences)
type	(*NX_CHAR*)		
wavelength	*NX_FLOAT*	*NX_WAVELENGTH*	
origin	*NXgeometry*		See table *NXsnshisto/NXentry/NXinstrument/NXcrysta Members*.

NXsnshisto/NXentry/NXinstrument/NXdetector/NXgeometry Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXdetector/NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
orientation	*NXorientation*		See table *NXsnshisto/NXentry/NXinstrument/NXdetect Members*.
shape	*NXshape*		See table *NXsnshisto/NXentry/NXinstrument/NXdetect Members*.
translation	*NXtranslation*		See table *NXsnshisto/NXentry/NXinstrument/NXdetect Members*.

NXsnshisto/NXentry/NXinstrument/NXaperture/NXgeometry Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXaperture/NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
orientation	*NXorientation*		See table *NXsnshisto/NXentry/NXinstrument/NXapertu Members*.
shape	*NXshape*		See table *NXsnshisto/NXentry/NXinstrument/NXapertu Members*.
translation	*NXtranslation*		See table *NXsnshisto/NXentry/NXinstrument/NXapertu Members*.

NXsnshisto/NXentry/NXinstrument/NXcrystal/NXgeometry Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXcrystal/NXgeometry* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		
orientation	*NXorientation*		See table *NXsnshisto/NXentry/NXinstrument/NXcrysta Members*.
shape	*NXshape*		See table *NXsnshisto/NXentry/NXinstrument/NXcrysta Members*.
translation	*NXtranslation*		See table *NXsnshisto/NXentry/NXinstrument/NXcrysta Members*.

NXsnshisto/NXentry/NXinstrument/NXdetector/NXgeometry/NXorientation Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXdetector/NXgeometry/NXorientation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	*NX_FLOAT*		Six out of nine rotation parameters. Dimensions: (rank=1) • 1: 6

NXsnshisto/NXentry/NXinstrument/NXdetector/NXgeometry/NXshape Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXdetector/NXgeometry/NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		
shape	(*NX_CHAR*)		
size	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: 3

NXsnshisto/NXentry/NXinstrument/NXdetector/NXgeometry/NXtranslation Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXdetector/NXgeometry/NXtranslation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: 3

NXsnshisto/NXentry/NXinstrument/NXaperture/NXgeometry/NXorientation Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXaperture/NXgeometry/NXorientation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	*NX_FLOAT*		Six out of nine rotation parameters. Dimensions: (rank=1) • 1: 6

NXsnshisto/NXentry/NXinstrument/NXaperture/NXgeometry/NXshape Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXaperture/NXgeometry/NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		
shape	(*NX_CHAR*)		
size	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: 3

NXsnshisto/NXentry/NXinstrument/NXaperture/NXgeometry/NXtranslation Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXaperture/NXgeometry/NXtranslation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: 3

NXsnshisto/NXentry/NXinstrument/NXcrystal/NXgeometry/NXorientation Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXcrystal/NXgeometry/NXorientation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	*NX_FLOAT*		Six out of nine rotation parameters. Dimensions: (rank=1) • 1: 6

NXsnshisto/NXentry/NXinstrument/NXcrystal/NXgeometry/NXshape Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXcrystal/NXgeometry/NXshape* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	(*NX_CHAR*)		
shape	(*NX_CHAR*)		
size	*NX_FLOAT*	*NX_LENGTH*	

NXsnshisto/NXentry/NXinstrument/NXcrystal/NXgeometry/NXtranslation Members Declarations in the *NXsnshisto/NXentry/NXinstrument/NXcrystal/NXgeometry/NXtranslation* group.

Name and Attributes	Type	Units	Description (and Occurrences)
distance	*NX_FLOAT*	*NX_LENGTH*	Dimensions: (rank=1) • 1: 3

NXsolenoid_magnet

version	category	extends	groups cited
1.0	contributed	*NXobject*	*NXlog*

definition for a solenoid magnet.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXsolenoid_magnet.nxdl.xml

svnid: Id: NXsolenoid_magnet.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXsolenoid_magnet

```
NXsolenoid_magnet (contributed definition, version 1.0)
  (base class definition, NXentry or NXsubentry not found)
  beamline_distance:NX_FLOAT
  description:NX_CHAR
  set_current:NX_FLOAT
  read_current:NXlog
    value:NX_CHAR
  read_voltage:NXlog
    value:NX_CHAR
```

NXsolenoid_magnet Members Declarations in the *NXsolenoid_magnet* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	*NX_CHAR*		extended description of the magnet.
beamline_distance	*NX_FLOAT*	*NX_LENGTH*	define position of beamline element relative to production target
set_current	*NX_FLOAT*	*NX_CURRENT*	current set on supply.
read_current	*NXlog*		current read from supply. See table *NXsolenoid_magnet/NXlog Members*.
read_voltage	*NXlog*		voltage read from supply. See table *NXsolenoid_magnet/NXlog Members*.

NXsolenoid_magnet/NXlog Members Declarations in the *NXsolenoid_magnet/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_CURRENT*	

NXsolenoid_magnet/NXlog Members Declarations in the *NXsolenoid_magnet/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_VOLTAGE*	

NXspin_rotator

version	category	extends	groups cited
1.0	contributed	*NXobject*	*NXlog*

definition for a spin rotator.

symbol list: No symbol table.

***NXDL* source:** http://svn.nexusformat.org/definitions/trunk/contributed_definitions/NXspin_rotator.nxdl.xml

svnid: Id: NXspin_rotator.nxdl.xml 1060 2012-03-06 15:00:33Z Pete Jemian

Structure of NXspin_rotator

```
NXspin_rotator (contributed definition, version 1.0)
  (base class definition, NXentry or NXsubentry not found)
  beamline_distance:NX_FLOAT
  description:NX_CHAR
  set_Bfield_current:NX_FLOAT
  set_Efield_voltage:NX_FLOAT
  read_Bfield_current:NXlog
    value:NX_CHAR
  read_Bfield_voltage:NXlog
    value:NX_CHAR
  read_Efield_current:NXlog
    value:NX_CHAR
  read_Efield_voltage:NXlog
    value:NX_CHAR
```

NXspin_rotator Members Declarations in the *NXspin_rotator* group.

Name and Attributes	Type	Units	Description (and Occurrences)
description	*NX_CHAR*		extended description of the spin rotator.
beamline_distance	*NX_FLOAT*	*NX_LENGTH*	define position of beamline element relative to production target
set_Bfield_current	*NX_FLOAT*	*NX_CURRENT*	current set on magnet supply.
set_Efield_voltage	*NX_FLOAT*	*NX_VOLTAGE*	current set on HT supply.
read_Bfield_current	*NXlog*		current read from magnet supply. See table *NXspin_rotator/NXlog Members*.
read_Bfield_voltage	*NXlog*		voltage read from magnet supply. See table *NXspin_rotator/NXlog Members*.
read_Efield_current	*NXlog*		current read from HT supply. See table *NXspin_rotator/NXlog Members*.
read_Efield_voltage	*NXlog*		voltage read from HT supply. See table *NXspin_rotator/NXlog Members*.

NXspin_rotator/NXlog Members Declarations in the *NXspin_rotator/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_CURRENT*	

NXspin_rotator/NXlog Members Declarations in the *NXspin_rotator/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_VOLTAGE*	

NXspin_rotator/NXlog Members Declarations in the *NXspin_rotator/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_CURRENT*	

NXspin_rotator/NXlog Members Declarations in the *NXspin_rotator/NXlog* group.

Name and Attributes	Type	Units	Description (and Occurrences)
value	(*NX_CHAR*)	*NX_VOLTAGE*	

2.4 Examples of writing and reading NeXus data files

Simple examples of reading and writing NeXus data files are provided in the *NeXus Introduction* chapter and also in the *NAPI: NeXus Application Programmer Interface* chapter. Here, three examples are provided showing how to write a NeXus data file without using the NAPI.

2.4.1 Code Examples that use the NAPI

Various examples are given that show how to read and write NeXus data files using the *NAPI: NeXus Application Programmer Interface*.

Example NeXus programs using NAPI

NAPI Simple 2-D Write Example (C, F77, F90)

Code examples are provided in this section that write 2-D data to a NeXus HDF5 file in C, F77, and F90 languages using the NAPI.

The following code reads a two-dimensional set `counts` with dimension scales of `t` and `phi` using local routines, and then writes a NeXus file containing a single `NXentry` group and a single `NXdata` group. This is the simplest data file that conforms to the NeXus standard. The same code is provided in C, F77, and F90 versions. Compare these code examples with *Example NeXus C programs using native HDF5 commands*.

NAPI C Example: write simple NeXus file

```
/* $Id: napi-example.c 1091 2012-05-28 21:10:09Z Pete Jemian $ */

#include "napi.h"

int main()
{
    int counts[50][1000], n_t=1000, n_p=50, dims[2], i;
    float t[1000], phi[50];
    NXhandle file_id;
/*
 * Read in data using local routines to populate phi and counts
 *
 * for example you may create a getdata() function and call
 *
 *      getdata (n_t, t, n_p, phi, counts);
 */
/* Open output file and output global attributes */
    NXopen ("NXfile.nxs", NXACC_CREATE5, &file_id);
      NXputattr (file_id, "user_name", "Joe Bloggs", 10, NX_CHAR);
/* Open top-level NXentry group */
      NXmakegroup (file_id, "Entry1", "NXentry");
      NXopengroup (file_id, "Entry1", "NXentry");
/* Open NXdata group within NXentry group */
        NXmakegroup (file_id, "Data1", "NXdata");
        NXopengroup (file_id, "Data1", "NXdata");
/* Output time channels */
          NXmakedata (file_id, "time_of_flight", NX_FLOAT32, 1, &n_t);
          NXopendata (file_id, "time_of_flight");
            NXputdata (file_id, t);
```

```
            NXputattr (file_id, "units", "microseconds", 12, NX_CHAR);
          NXclosedata (file_id);
/* Output detector angles */
          NXmakedata (file_id, "polar_angle", NX_FLOAT32, 1, &n_p);
          NXopendata (file_id, "polar_angle");
            NXputdata (file_id, phi);
            NXputattr (file_id, "units", "degrees", 7, NX_CHAR);
          NXclosedata (file_id);
/* Output data */
          dims[0] = n_t;
          dims[1] = n_p;
          NXmakedata (file_id, "counts", NX_INT32, 2, dims);
          NXopendata (file_id, "counts");
            NXputdata (file_id, counts);
            i = 1;
            NXputattr (file_id, "signal", &i, 1, NX_INT32);
            NXputattr (file_id, "axes",  "polar_angle:time_of_flight", 26, NX_CHAR);
          NXclosedata (file_id);
/* Close NXentry and NXdata groups and close file */
        NXclosegroup (file_id);
      NXclosegroup (file_id);
    NXclose (&file_id);
    return;
}
```

NAPI F77 Example: write simple NeXus file

Note: The F77 interface is no longer being developed.

```
! $Id: napi-example.f77 552 2010-04-19 22:24:42Z Pete Jemian $

      program WRITEDATA

      include 'NAPIF.INC'
      integer*4 status, file_id(NXHANDLESIZE), counts(1000,50), n_p, n_t, dims(2)
      real*4 t(1000), phi(50)

!Read in data using local routines
      call getdata (n_t, t, n_p, phi, counts)
!Open output file
      status = NXopen ('NXFILE.NXS', NXACC_CREATE, file_id)
        status = NXputcharattr
     +         (file_id, 'user', 'Joe Bloggs', 10, NX_CHAR)
!Open top-level NXentry group
        status = NXmakegroup (file_id, 'Entry1', 'NXentry')
        status = NXopengroup (file_id, 'Entry1', 'NXentry')
!Open NXdata group within NXentry group
          status = NXmakegroup (file_id, 'Data1', 'NXdata')
          status = NXopengroup (file_id, 'Data1', 'NXdata')
!Output time channels
            status = NXmakedata
     +         (file_id, 'time_of_flight', NX_FLOAT32, 1, n_t)
            status = NXopendata (file_id, 'time_of_flight')
              status = NXputdata (file_id, t)
              status = NXputcharattr
     +         (file_id, 'units', 'microseconds', 12, NX_CHAR)
            status = NXclosedata (file_id)
!Output detector angles
```

```
            status = NXmakedata (file_id, 'polar_angle', NX_FLOAT32, 1, n_p)
            status = NXopendata (file_id, 'polar_angle')
              status = NXputdata (file_id, phi)
              status = NXputcharattr (file_id, 'units', 'degrees', 7, NX_CHAR)
            status = NXclosedata (file_id)
!Output data
            dims(1) = n_t
            dims(2) = n_p
            status = NXmakedata (file_id, 'counts', NX_INT32, 2, dims)
            status = NXopendata (file_id, 'counts')
              status = NXputdata (file_id, counts)
              status = NXputattr (file_id, 'signal', 1, 1, NX_INT32)
              status = NXputattr
     +          (file_id, 'axes', 'polar_angle:time_of_flight', 26, NX_CHAR)
            status = NXclosedata (file_id)
!Close NXdata and NXentry groups and close file
          status = NXclosegroup (file_id)
        status = NXclosegroup (file_id)
      status = NXclose (file_id)

      stop
      end
```

NAPI F90 Example: write simple NeXus file

```
! $Id: napi-example.f90 552 2010-04-19 22:24:42Z Pete Jemian $

program WRITEDATA

   use NXUmodule

   type(NXhandle) :: file_id
   integer, pointer :: counts(:,:)
   real, pointer :: t(:), phi(:)

!Use local routines to allocate pointers and fill in data
   call getlocaldata (t, phi, counts)
!Open output file
   if (NXopen ("NXfile.nxs", NXACC_CREATE, file_id) /= NX_OK) stop
   if (NXUwriteglobals (file_id, user="Joe Bloggs") /= NX_OK) stop
!Set compression parameters
   if (NXUsetcompress (file_id, NX_COMP_LZW, 1000) /= NX_OK) stop
!Open top-level NXentry group
   if (NXUwritegroup (file_id, "Entry1", "NXentry") /= NX_OK) stop
   !Open NXdata group within NXentry group
      if (NXUwritegroup (file_id, "Data1", "NXdata") /= NX_OK) stop
   !Output time channels
         if (NXUwritedata (file_id, "time_of_flight", t, "microseconds") /= NX_OK) stop
   !Output detector angles
         if (NXUwritedata (file_id, "polar_angle", phi, "degrees") /= NX_OK) stop
   !Output data
         if (NXUwritedata (file_id, "counts", counts, "counts") /= NX_OK) stop
            if (NXputattr (file_id, "signal", 1) /= NX_OK) stop
            if (NXputattr (file_id, "axes", "polar_angle:time_of_flight") /= NX_OK) stop
   !Close NXdata group
      if (NXclosegroup (file_id) /= NX_OK) stop
!Close NXentry group
   if (NXclosegroup (file_id) /= NX_OK) stop
```

```
!Close NeXus file
   if (NXclose (file_id) /= NX_OK) stop

end program WRITEDATA
```

NAPI Python Simple 3-D Write Example

A single code example is provided in this section that writes 3-D data to a NeXus HDF5 file in the Python language using the NAPI. The data file may be retrieved from the repository of NeXus data file examples:

data http://svn.nexusformat.org/definitions/exampledata/simple3D.h5

The data to be written to the file is a simple three-dimensional array (2 x 3 x 4) of integers. The single dataset is intended to demonstrate the order in which each value of the array is stored in a NeXus HDF5 data file.

NAPI Python Example: write simple NeXus file

```
#!/usr/bin/python

import sys
import nxs
import numpy

nf = nxs.open("simple3D.h5", "w5")

nf.makegroup("entry","NXentry")
nf.opengroup("entry","NXentry")

nf.makegroup("data","NXdata")
nf.opengroup("data","NXdata")

a = numpy.zeros((2,3,4),dtype=numpy.int)
val = 0
for i in range(2):
    for j in range(3):
        for k in range(4):
            a[i,j,k] = val
            val = val + 1

nf.makedata("test",'int32',[2,3,4])
nf.opendata("test")
nf.putdata(a)
nf.putattr("signal",1)
nf.closedata()

nf.closegroup() # NXdata
nf.closegroup() # NXentry

nf.close()

exit
```

View a NeXus HDF5 file using *h5dump*

For the purposes of an example, it is instructive to view the content of the NeXus HDF5 file produced by the above program. Since HDF5 is a binary file format, we cannot show the contents of the file directly in this manual. Instead,

we first we view the content by showing the output from the `h5dump` tool provided as part of the HDF5 tool kit: `h5dump simple3D.h5`

NAPI Python Example: `h5dump` output of NeXus HDF5 file

```
HDF5 "simple3D.h5" {
GROUP "/" {
   ATTRIBUTE "NeXus_version" {
      DATATYPE  H5T_STRING {
            STRSIZE 5;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "4.1.0"
      }
   }
   ATTRIBUTE "file_name" {
      DATATYPE  H5T_STRING {
            STRSIZE 11;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "simple3D.h5"
      }
   }
   ATTRIBUTE "HDF5_Version" {
      DATATYPE  H5T_STRING {
            STRSIZE 5;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "1.6.6"
      }
   }
   ATTRIBUTE "file_time" {
      DATATYPE  H5T_STRING {
            STRSIZE 24;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SCALAR
      DATA {
      (0): "2011-11-18 17:26:27+0100"
      }
   }
   GROUP "entry" {
      ATTRIBUTE "NX_class" {
         DATATYPE  H5T_STRING {
```

```
                STRSIZE 7;
                STRPAD H5T_STR_NULLTERM;
                CSET H5T_CSET_ASCII;
                CTYPE H5T_C_S1;
             }
          DATASPACE  SCALAR
          DATA {
          (0): "NXentry"
          }
       }
       GROUP "data" {
          ATTRIBUTE "NX_class" {
             DATATYPE  H5T_STRING {
                   STRSIZE 6;
                   STRPAD H5T_STR_NULLTERM;
                   CSET H5T_CSET_ASCII;
                   CTYPE H5T_C_S1;
                }
             DATASPACE  SCALAR
             DATA {
             (0): "NXdata"
             }
          }
          DATASET "test" {
             DATATYPE  H5T_STD_I32LE
             DATASPACE  SIMPLE { ( 2, 3, 4 ) / ( 2, 3, 4 ) }
             DATA {
             (0,0,0): 0, 1, 2, 3,
             (0,1,0): 4, 5, 6, 7,
             (0,2,0): 8, 9, 10, 11,
             (1,0,0): 12, 13, 14, 15,
             (1,1,0): 16, 17, 18, 19,
             (1,2,0): 20, 21, 22, 23
             }
             ATTRIBUTE "signal" {
                DATATYPE  H5T_STD_I32LE
                DATASPACE  SCALAR
                DATA {
                (0): 1
                }
             }
          }
       }
    }
}
}
```

View a NeXus HDF5 file using *h5toText.py*

The output of `h5dump` contains a lot of structural information about the HDF5 file that can distract us from the actual content we added to the file. Next, we show the output from a custom Python tool (`h5toText.py`) that we describe in a later section (*h5toText support module*) of this chapter. This tool was developed to show the actual data content of an HDF5 file that we create.

NAPI Python Example: `h5toText` output of NeXus HDF5 file

```
simple3D.h5:NeXus data file
  @NeXus_version = 4.1.0
  @file_name = simple3D.h5
  @HDF5_Version = 1.6.6
  @file_time = 2011-11-18 17:26:27+0100
  entry:NXentry
    @NX_class = NXentry
    data:NXdata
      @NX_class = NXdata
      test:NX_INT32[2,3,4] = __array
        @signal = 1
        __array = [
            [
              [0, 1, 2, 3]
              [4, 5, 6, 7]
              [8, 9, 10, 11]
            ]
            [
              [12, 13, 14, 15]
              [16, 17, 18, 19]
              [20, 21, 22, 23]
            ]
          ]
```

2.4.2 Code Examples that do not use the NAPI

Sometimes, for whatever reason, it is necessary to write or read NeXus files without using the routines provided by the *NAPI: NeXus Application Programmer Interface*. Each example in this section is written to support just one of the low-level file formats supported by NeXus (HDF4, HDF5, or XML).

Example NeXus C programs using native HDF5 commands

C-language code examples are provided for writing and reading NeXus-compliant files using the native HDF5 interfaces. These examples are derived from the simple NAPI examples for *writing* and *reading* given in the *Introduction* chapter. Compare these code examples with *Example NeXus programs using NAPI*.

Writing a simple NeXus file using native HDF5 commands in C

```
/**
 * This is an example how to write a valid NeXus file
 * using the HDF-5 API alone. Ths structure which is
 * going to be created is:
 *
 * scan:NXentry
 *      data:NXdata
 *         counts[]
 *            @signal=1
 *         two_theta[]
 *            @units=degrees
 *
 *  WARNING: each of the HDF function below needs to be
 *  wrapped into something like:
 *
 *  if((hdfid = H5function(...)) < 0){
```

```
 *     handle error gracefully
 *  }
 *  I left the error checking out in order to keep the
 *  code clearer
 *
 *  This also installs a link from /scan/data/two_theta to /scan/hugo
 *
 *  Mark Koennecke, October 2011
 */
#include <hdf5.h>
#include <stdlib.h>
#include <string.h>

#define LENGTH 400
int main(int argc, char *argv[])
{
  float two_theta[LENGTH];
  int counts[LENGTH], i, rank, signal;

  /* HDF-5 handles */
  hid_t fid, fapl, gid, atts, atttype, attid;
  hid_t datatype, dataspace, dataprop, dataid;
  hsize_t dim[1], maxdim[1];


  /* create some data: nothing NeXus or HDF-5 specific */
  for(i = 0; i < LENGTH; i++){
    two_theta[i] = 10. + .1*i;
    counts[i] = (int)(1000 * ((float)random()/(float)RAND_MAX));
  }
  dim[0] = LENGTH;
  maxdim[0] = LENGTH;
  rank = 1;



  /*
   * open the file. The file attribute forces normal file
   * closing behaviour down HDF-5's throat
   */
  fapl = H5Pcreate(H5P_FILE_ACCESS);
  H5Pset_fclose_degree(fapl,H5F_CLOSE_STRONG);
  fid = H5Fcreate("NXfile.h5", H5F_ACC_TRUNC, H5P_DEFAULT,fapl);
  H5Pclose(fapl);


  /*
   * create scan:NXentry
   */
  gid = H5Gcreate(fid, (const char *)"scan",0);
  /*
   * store the NX_class attribute. Notice that you
   * have to take care to close those hids after use
   */
  atts = H5Screate(H5S_SCALAR);
  atttype = H5Tcopy(H5T_C_S1);
  H5Tset_size(atttype, strlen("NXentry"));
  attid = H5Acreate(gid,"NX_class", atttype, atts, H5P_DEFAULT);
```

```
  H5Awrite(attid, atttype, (char *)"NXentry");
  H5Sclose(atts);
  H5Tclose(atttype);
  H5Aclose(attid);

  /*
   * same thing for data:Nxdata in scan:NXentry.
   * A subroutine would be nice to have here.......
   */
  gid = H5Gcreate(fid, (const char *)"/scan/data",0);
  atts = H5Screate(H5S_SCALAR);
  atttype = H5Tcopy(H5T_C_S1);
  H5Tset_size(atttype, strlen("NXdata"));
  attid = H5Acreate(gid,"NX_class", atttype, atts, H5P_DEFAULT);
  H5Awrite(attid, atttype, (char *)"NXdata");
  H5Sclose(atts);
  H5Tclose(atttype);
  H5Aclose(attid);

  /*
   * store the counts dataset
   */
  dataspace = H5Screate_simple(rank,dim,maxdim);
  datatype = H5Tcopy(H5T_NATIVE_INT);
  dataprop = H5Pcreate(H5P_DATASET_CREATE);
  dataid = H5Dcreate(gid,(char *)"counts",datatype,dataspace,dataprop);
  H5Dwrite(dataid, datatype, H5S_ALL, H5S_ALL, H5P_DEFAULT, counts);
  H5Sclose(dataspace);
  H5Tclose(datatype);
  H5Pclose(dataprop);
  /*
   * set the signal=1 attribute
   */
  atts = H5Screate(H5S_SCALAR);
  atttype = H5Tcopy(H5T_NATIVE_INT);
  H5Tset_size(atttype,1);
  attid = H5Acreate(dataid,"signal", atttype, atts, H5P_DEFAULT);
  signal = 1;
  H5Awrite(attid, atttype, &signal);
  H5Sclose(atts);
  H5Tclose(atttype);
  H5Aclose(attid);

  H5Dclose(dataid);

  /*
   * store the two_theta dataset
   */
  dataspace = H5Screate_simple(rank,dim,maxdim);
  datatype = H5Tcopy(H5T_NATIVE_FLOAT);
  dataprop = H5Pcreate(H5P_DATASET_CREATE);
  dataid = H5Dcreate(gid,(char *)"two_theta",datatype,dataspace,dataprop);
  H5Dwrite(dataid, datatype, H5S_ALL, H5S_ALL, H5P_DEFAULT, two_theta);
  H5Sclose(dataspace);
  H5Tclose(datatype);
  H5Pclose(dataprop);

  /*
```

```
    * set the units attribute
    */
   atttype = H5Tcopy(H5T_C_S1);
   H5Tset_size(atttype, strlen("degrees"));
   atts = H5Screate(H5S_SCALAR);
   attid = H5Acreate(dataid,"units", atttype, atts, H5P_DEFAULT);
   H5Awrite(attid, atttype, (char *)"degrees");
   H5Sclose(atts);
   H5Tclose(atttype);
   H5Aclose(attid);
  /*
    * set the target attribute for linking
    */
   atttype = H5Tcopy(H5T_C_S1);
   H5Tset_size(atttype, strlen("/scan/data/two_theta"));
   atts = H5Screate(H5S_SCALAR);
   attid = H5Acreate(dataid,"target", atttype, atts, H5P_DEFAULT);
   H5Awrite(attid, atttype, (char *)"/scan/data/two_theta");
   H5Sclose(atts);
   H5Tclose(atttype);
   H5Aclose(attid);


   H5Dclose(dataid);

   /*
    * make a link in /scan to /scan/data/two_theta, thereby
    * renaming two_theta to hugo
    */
   H5Glink(fid,H5G_LINK_HARD,"/scan/data/two_theta","/scan/hugo");

   /*
    * close the file
    */
   H5Fclose(fid);
}
```

Reading a simple NeXus file using native HDF5 commands in C

```
/**
 * Reading example for reading NeXus files with plain
 * HDF-5 API calls. This reads out counts and two_theta
 * out of the file generated by nxh5write.
 *
 * WARNING: I left out all error checking in this example.
 * In production code you have to take care of those errors
 *
 * Mark Koennecke, October 2011
 */
#include <hdf5.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  float *two_theta = NULL;
  int *counts = NULL,  rank, i;
```

```
  hid_t fid, dataid, fapl;
  hsize_t *dim = NULL;
  hid_t datatype, dataspace, memdataspace;

  /*
   * Open file, thereby enforcing proper file close
   * semantics
   */
  fapl = H5Pcreate(H5P_FILE_ACCESS);
  H5Pset_fclose_degree(fapl,H5F_CLOSE_STRONG);
  fid = H5Fopen("NXfile.h5", H5F_ACC_RDONLY,fapl);
  H5Pclose(fapl);

  /*
   * open and read the counts dataset
   */
  dataid = H5Dopen(fid,"/scan/data/counts");
  dataspace = H5Dget_space(dataid);
  rank = H5Sget_simple_extent_ndims(dataspace);
  dim = malloc(rank*sizeof(hsize_t));
  H5Sget_simple_extent_dims(dataspace, dim, NULL);
  counts = malloc(dim[0]*sizeof(int));
  memdataspace = H5Tcopy(H5T_NATIVE_INT32);
  H5Dread(dataid,memdataspace,H5S_ALL, H5S_ALL,H5P_DEFAULT, counts);
  H5Dclose(dataid);
  H5Sclose(dataspace);
  H5Tclose(memdataspace);

  /*
   * open and read the two_theta data set
   */
  dataid = H5Dopen(fid,"/scan/data/two_theta");
  dataspace = H5Dget_space(dataid);
  rank = H5Sget_simple_extent_ndims(dataspace);
  dim = malloc(rank*sizeof(hsize_t));
  H5Sget_simple_extent_dims(dataspace, dim, NULL);
  two_theta = malloc(dim[0]*sizeof(float));
  memdataspace = H5Tcopy(H5T_NATIVE_FLOAT);
  H5Dread(dataid,memdataspace,H5S_ALL, H5S_ALL,H5P_DEFAULT, two_theta);
  H5Dclose(dataid);
  H5Sclose(dataspace);
  H5Tclose(memdataspace);



  H5Fclose(fid);

  for(i = 0; i < dim[0]; i++){
    printf("%8.2f %10d\n", two_theta[i], counts[i]);
  }

}
```

Python Examples using `h5py`

One way to gain a quick familiarity with NeXus is to start working with some data. For at least the first few examples in this section, we have a simple two-column set of 1-D data, collected as part of a series of alignment scans by the

APS USAXS instrument during the time it was stationed at beam line 32ID. We will show how to write this data using the Python language and the `h5py` package [4] (using `h5py` calls directly rather than using the NeXus NAPI). The actual data to be written was extracted (elsewhere) from a `spec` [5] data file and read as a text block from a file by the Python source code. Our examples will start with the simplest case and add only mild complexity with each new case since these examples are meant for those who are unfamiliar with NeXus.

The data shown plotted in the next figure will be written to the NeXus HDF5 file using the only two required NeXus objects `NXentry` and `NXdata` in the first example and then minor variations on this structure in the next two examples. The data model is identical to the one in the *Introduction* chapter except that the names will be different, as shown below:

Figure 2.10: data structure, (from Introduction)

our h5py example

```
/entry:NXentry
    /mr_scan:NXdata
       /mr : float64[31]
       /I00 : int32[31]
```

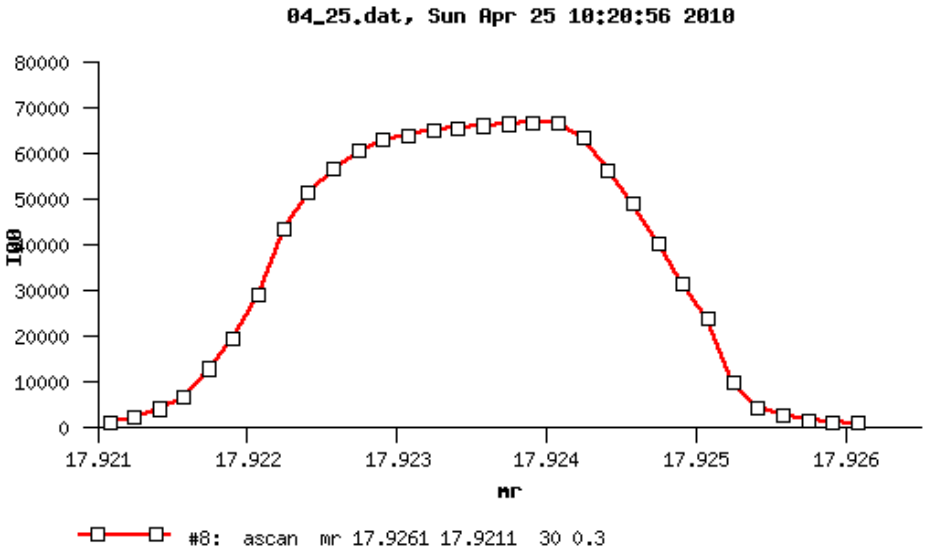


Figure 2.11: plot of our *mr_scan*

[4] *h5py*: http://code.google.com/p/h5py
[5] *SPEC*: http://certif.com/spec.html

two-column data for our *mr_scan*

```
17.92608    1037
17.92591    1318
17.92575    1704
17.92558    2857
17.92541    4516
17.92525    9998
17.92508    23819
17.92491    31662
17.92475    40458
17.92458    49087
17.92441    56514
17.92425    63499
17.92408    66802
17.92391    66863
17.92375    66599
17.92358    66206
17.92341    65747
17.92325    65250
17.92308    64129
17.92291    63044
17.92275    60796
17.92258    56795
17.92241    51550
17.92225    43710
17.92208    29315
17.92191    19782
17.92175    12992
17.92158    6622
17.92141    4198
17.92125    2248
17.92108    1321
```

Writing the simplest data using `h5py`

These two examples show how to write the simplest data (above). One example writes the data directly to the *NXdata* group while the other example writes the data to `NXinstrument/NXdetector/data` and then creates a soft link to that data in `NXdata`.

`h5py` example writing the simplest NeXus data file In this example, the 1-D scan data will be written into the simplest possible NeXus HDF5 data file, containing only the required NeXus components. NeXus requires at least one *NXentry* group at the root level of an HDF5 file. The `NXentry` group contains *all the data and associated information that comprise a single measurement.* NeXus also requires that each `NXentry` group must contain at least one *NXdata* group. `NXdata` is used to describe the plottable data in the `NXentry` group. The simplest place to store data in a NeXus file is directly in the `NXdata` group, as shown in the next figure.

In the *above figure*, the data file (`writer_1_3_h5py.hdf5`) contains a hierarchy of items, starting with an `NXentry` named `entry`. (The full HDF5 path reference, `/entry` in this case, is shown to the right of each component in the data structure.) The next `h5py` code example will show how to build an HDF5 data file with this structure. Starting with the numerical data described above, the only information written to the file is the *absolute* minimum information NeXus requires. In this example, you can see how the HDF5 file is created, how *Data Groups* and datasets (*Data Fields*) are created, and how *Data Attributes* are assigned. Note particularly the `NX_class` attribute on each HDF5 group that describes which of the NeXus *Base Class Definitions* is being used. When the

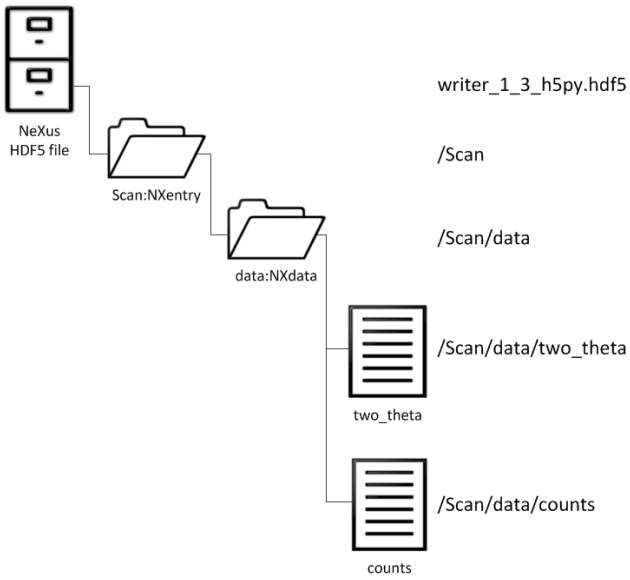


Figure 2.12: Simple Example

next Python program (`writer_1_3_h5py.py`) is run from the command line (and there are no problems), the `writer_1_3_h5py.hdf5` file is generated.

```
#!/usr/bin/env python
'''
Writes the simplest NeXus HDF5 file using h5py
according to the example from Figure 1.3
in the Introduction chapter
'''

import h5py
import numpy

INPUT_FILE = 'input.dat'
HDF5_FILE = 'writer_1_3_h5py.hdf5'

#---------------------------

tthData, countsData = numpy.loadtxt(INPUT_FILE).T

f = h5py.File(HDF5_FILE, "w")  # create the HDF5 NeXus file
# since this is a simple example, no attributes are used at this point

nxentry = f.create_group('Scan')
nxentry.attrs["NX_class"] = 'NXentry'

nxdata = nxentry.create_group('data')
nxdata.attrs["NX_class"] = 'NXdata'

tth = nxdata.create_dataset("two_theta", data=tthData)
tth.attrs['units'] = "degrees"

counts = nxdata.create_dataset("counts", data=countsData)
counts.attrs['units'] = "counts"
counts.attrs['signal'] = 1
```

```
counts.attrs['axes'] = "two_theta"

f.close()    # be CERTAIN to close the file
```

We wish to make things a bit simpler for ourselves when creating the common structures we use in our data files. To help, we gather together some of the common concepts such as *create a file*, *create a NeXus group*, *create a dataset* and start to build a helper library. (See *mylib support module* for more details.) Here, we call it `my_lib`. Applying it to the simple example above, our code only becomes a couple lines shorter! (Let's hope the library starts to help in larger or more complicated projects.) Here's the revision that replaces direct calls to `numpy` and `h5py` with calls to our library. It generates the file `writer_1_3.hdf5`.

```
#!/usr/bin/env python
'''
Writes the simplest NeXus HDF5 file using
a simple helper library with h5py and numpy calls
according to the example from Figure 1.3
in the Introduction chapter
'''

import my_lib

INPUT_FILE = 'input.dat'
HDF5_FILE = 'writer_1_3.hdf5'

#---------------------------

tthData, countsData = my_lib.get2ColumnData(INPUT_FILE)

f = my_lib.makeFile(HDF5_FILE)
# since this is a simple example, no attributes are used at this point

nxentry = my_lib.makeGroup(f, 'Scan', 'NXentry')
nxdata = my_lib.makeGroup(nxentry, 'data', 'NXdata')

my_lib.makeDataset(nxdata, "two_theta", tthData, units='degrees')
my_lib.makeDataset(nxdata, "counts", countsData,
                   units='counts', signal=1, axes='two_theta')

f.close()    # be CERTAIN to close the file
```

One of the tools provided with the HDF5 support libraries is the `h5dump` command, a command-line tool to print out the contents of an HDF5 data file. With no better tool in place (the output is verbose), this is a good tool to investigate what has been written to the HDF5 file. View this output from the command line using `h5dump writer_1_3.hdf5`. Compare the data contents with the numbers shown above. Note that the various HDF5 data types have all been decided by the `h5py` support package.

Note: The only difference between this file and one written using the NAPI is that the NAPI file will have some additional, optional attributes set at the root level of the file that tells the original file name, time it was written, and some version information about the software involved.

```
HDF5 "writer_1_3.hdf5" {
GROUP "/" {
   GROUP "Scan" {
      ATTRIBUTE "NX_class" {
         DATATYPE  H5T_STRING {
               STRSIZE 7;
```

```
                STRPAD H5T_STR_NULLPAD;
                CSET H5T_CSET_ASCII;
                CTYPE H5T_C_S1;
             }
          DATASPACE  SCALAR
          DATA {
          (0): "NXentry"
          }
       }
       GROUP "data" {
          ATTRIBUTE "NX_class" {
             DATATYPE  H5T_STRING {
                   STRSIZE 6;
                   STRPAD H5T_STR_NULLPAD;
                   CSET H5T_CSET_ASCII;
                   CTYPE H5T_C_S1;
                }
             DATASPACE  SCALAR
             DATA {
             (0): "NXdata"
             }
          }
          DATASET "counts" {
             DATATYPE  H5T_STD_I32LE
             DATASPACE  SIMPLE { ( 31 ) / ( 31 ) }
             DATA {
             (0): 1037, 1318, 1704, 2857, 4516, 9998, 23819, 31662, 40458,
             (9): 49087, 56514, 63499, 66802, 66863, 66599, 66206, 65747,
             (17): 65250, 64129, 63044, 60796, 56795, 51550, 43710, 29315,
             (25): 19782, 12992, 6622, 4198, 2248, 1321
             }
             ATTRIBUTE "units" {
                DATATYPE  H5T_STRING {
                      STRSIZE 6;
                      STRPAD H5T_STR_NULLPAD;
                      CSET H5T_CSET_ASCII;
                      CTYPE H5T_C_S1;
                   }
                DATASPACE  SCALAR
                DATA {
                (0): "counts"
                }
             }
             ATTRIBUTE "signal" {
                DATATYPE  H5T_STRING {
                      STRSIZE 1;
                      STRPAD H5T_STR_NULLPAD;
                      CSET H5T_CSET_ASCII;
                      CTYPE H5T_C_S1;
                   }
                DATASPACE  SCALAR
                DATA {
                (0): "1"
                }
             }
             ATTRIBUTE "axes" {
                DATATYPE  H5T_STRING {
                      STRSIZE 9;
```

```
                        STRPAD H5T_STR_NULLPAD;
                        CSET H5T_CSET_ASCII;
                        CTYPE H5T_C_S1;
                     }
                  DATASPACE  SCALAR
                  DATA {
                  (0): "two_theta"
                  }
               }
            }
            DATASET "two_theta" {
               DATATYPE  H5T_IEEE_F64LE
               DATASPACE  SIMPLE { ( 31 ) / ( 31 ) }
               DATA {
               (0): 17.9261, 17.9259, 17.9258, 17.9256, 17.9254, 17.9252,
               (6): 17.9251, 17.9249, 17.9247, 17.9246, 17.9244, 17.9243,
               (12): 17.9241, 17.9239, 17.9237, 17.9236, 17.9234, 17.9232,
               (18): 17.9231, 17.9229, 17.9228, 17.9226, 17.9224, 17.9222,
               (24): 17.9221, 17.9219, 17.9217, 17.9216, 17.9214, 17.9213,
               (30): 17.9211
               }
               ATTRIBUTE "units" {
                  DATATYPE  H5T_STRING {
                        STRSIZE 7;
                        STRPAD H5T_STR_NULLPAD;
                        CSET H5T_CSET_ASCII;
                        CTYPE H5T_C_S1;
                     }
                  DATASPACE  SCALAR
                  DATA {
                  (0): "degrees"
                  }
               }
            }
         }
      }
}
}
```

Since the output of `h5dump` is verbose, a tool (see *h5toText support module*) was created to print out the structure of HDF5 data files. This tool provides a simplified view of the NeXus file. It is run with a command like this: `python h5toText.py h5dump writer_1_3.hdf5`. Here is the output:

```
writer_1_3.hdf5:NeXus data file
  Scan:NXentry
    @NX_class = NXentry
    data:NXdata
      @NX_class = NXdata
      counts:NX_INT32[31] = __array
        @units = counts
        @signal = 1
        @axes = two_theta
        __array = [1037, 1318, 1704, '...', 1321]
      two_theta:NX_FLOAT64[31] = __array
        @units = degrees
        __array = [17.926079999999999, 17.925909999999998,
                   17.925750000000001, '...', 17.92108]
```

As the data files in these examples become more complex, you will appreciate the information density provided by the

`h5toText.py` tool.

h5py example writing a simple NeXus data file with links Building on the previous example, we wish to identify our measured data with the detector on the instrument where it was generated. In this hypothetical case, since the detector was positioned at some angle *two_theta*, we choose to store both datasets, `two_theta` and `counts`, in a NeXus group. One appropriate NeXus group is *NXdetector*. This group is placed in a *NXinstrument* group which is placed in a *NXentry* group. Still, NeXus requires a *NXdata* group. Rather than duplicate the same data already placed in the detector group, we choose to link to those datasets from the `NXdata` group. (Compare the next figure with *Linking in a NeXus file* in the *NeXus Design* chapter of the NeXus User Manual.) The *NeXus Design* chapter provides a figure (*Linking in a NeXus file*) with a small variation from our previous example, placing the measured data within the `/entry/instrument/detector` group. Links are made from that data to the `/entry/data` group.

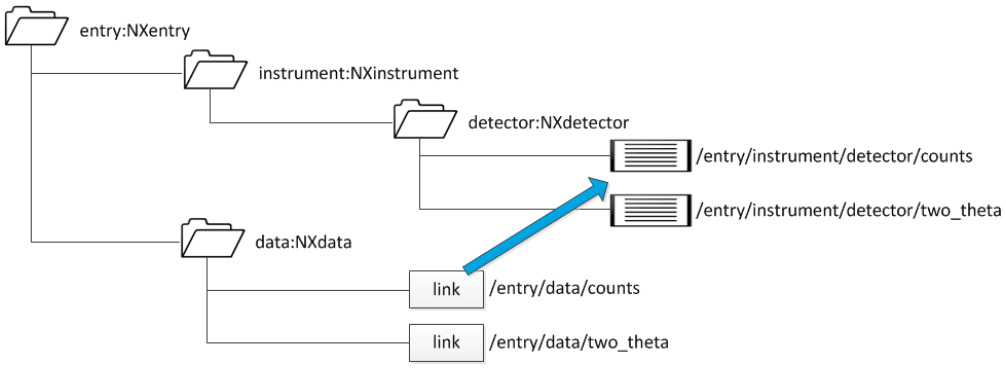


Figure 2.13: h5py example showing linking in a NeXus file

The Python code to build an HDF5 data file with that structure (using numerical data from the previous example) is shown below.

```
#!/usr/bin/env python
'''
Writes a simple NeXus HDF5 file using h5py with links
according to the example from Figure 2.1 in the Design chapter
'''

import my_lib

INPUT_FILE = 'input.dat'
HDF5_FILE = 'writer_2_1.hdf5'

#---------------------------

tthData, countsData = my_lib.get2ColumnData(INPUT_FILE)

f = my_lib.makeFile(HDF5_FILE)  # create the HDF5 NeXus file

nxentry = my_lib.makeGroup(f, 'entry', 'NXentry')
nxinstrument = my_lib.makeGroup(nxentry, 'instrument', 'NXinstrument')
nxdetector = my_lib.makeGroup(nxinstrument, 'detector', 'NXdetector')

tth = my_lib.makeDataset(nxdetector, "two_theta", tthData, units='degrees')
counts = my_lib.makeDataset(nxdetector, "counts", countsData,
                    units='counts', signal=1, axes='two_theta')

nxdata = my_lib.makeGroup(nxentry, 'data', 'NXdata')
```

```
my_lib.makeLink(nxdetector, tth, nxdata.name+'/two_theta')
my_lib.makeLink(nxdetector, counts, nxdata.name+'/counts')

f.close()    # be CERTAIN to close the file
```

It is interesting to compare the output of the `h5dump` of the data file `writer_2_1.hdf5` with our Python instructions.

```
HDF5 "writer_2_1.hdf5" {
GROUP "/" {
   GROUP "entry" {
      ATTRIBUTE "NX_class" {
         DATATYPE  H5T_STRING {
               STRSIZE 7;
               STRPAD H5T_STR_NULLPAD;
               CSET H5T_CSET_ASCII;
               CTYPE H5T_C_S1;
            }
         DATASPACE  SCALAR
         DATA {
         (0): "NXentry"
         }
      }
      GROUP "data" {
         ATTRIBUTE "NX_class" {
            DATATYPE  H5T_STRING {
                  STRSIZE 6;
                  STRPAD H5T_STR_NULLPAD;
                  CSET H5T_CSET_ASCII;
                  CTYPE H5T_C_S1;
               }
            DATASPACE  SCALAR
            DATA {
            (0): "NXdata"
            }
         }
         DATASET "counts" {
            DATATYPE  H5T_STD_I32LE
            DATASPACE  SIMPLE { ( 31 ) / ( 31 ) }
            DATA {
            (0): 1037, 1318, 1704, 2857, 4516, 9998, 23819, 31662, 40458,
            (9): 49087, 56514, 63499, 66802, 66863, 66599, 66206, 65747,
            (17): 65250, 64129, 63044, 60796, 56795, 51550, 43710, 29315,
            (25): 19782, 12992, 6622, 4198, 2248, 1321
            }
            ATTRIBUTE "units" {
               DATATYPE  H5T_STRING {
                     STRSIZE 6;
                     STRPAD H5T_STR_NULLPAD;
                     CSET H5T_CSET_ASCII;
                     CTYPE H5T_C_S1;
                  }
               DATASPACE  SCALAR
               DATA {
               (0): "counts"
               }
            }
            ATTRIBUTE "signal" {
```

```
            DATATYPE  H5T_STRING {
                  STRSIZE 1;
                  STRPAD H5T_STR_NULLPAD;
                  CSET H5T_CSET_ASCII;
                  CTYPE H5T_C_S1;
               }
            DATASPACE  SCALAR
            DATA {
            (0): "1"
            }
         }
         ATTRIBUTE "axes" {
            DATATYPE  H5T_STRING {
                  STRSIZE 9;
                  STRPAD H5T_STR_NULLPAD;
                  CSET H5T_CSET_ASCII;
                  CTYPE H5T_C_S1;
               }
            DATASPACE  SCALAR
            DATA {
            (0): "two_theta"
            }
         }
         ATTRIBUTE "target" {
            DATATYPE  H5T_STRING {
                  STRSIZE 33;
                  STRPAD H5T_STR_NULLPAD;
                  CSET H5T_CSET_ASCII;
                  CTYPE H5T_C_S1;
               }
            DATASPACE  SCALAR
            DATA {
            (0): "/entry/instrument/detector/counts"
            }
         }
      }
      DATASET "two_theta" {
         DATATYPE  H5T_IEEE_F64LE
         DATASPACE  SIMPLE { ( 31 ) / ( 31 ) }
         DATA {
         (0): 17.9261, 17.9259, 17.9258, 17.9256, 17.9254, 17.9252,
         (6): 17.9251, 17.9249, 17.9247, 17.9246, 17.9244, 17.9243,
         (12): 17.9241, 17.9239, 17.9237, 17.9236, 17.9234, 17.9232,
         (18): 17.9231, 17.9229, 17.9228, 17.9226, 17.9224, 17.9222,
         (24): 17.9221, 17.9219, 17.9217, 17.9216, 17.9214, 17.9213,
         (30): 17.9211
         }
         ATTRIBUTE "units" {
            DATATYPE  H5T_STRING {
                  STRSIZE 7;
                  STRPAD H5T_STR_NULLPAD;
                  CSET H5T_CSET_ASCII;
                  CTYPE H5T_C_S1;
               }
            DATASPACE  SCALAR
            DATA {
            (0): "degrees"
            }
```

```
109             }
110             ATTRIBUTE "target" {
111                DATATYPE  H5T_STRING {
112                      STRSIZE 36;
113                      STRPAD H5T_STR_NULLPAD;
114                      CSET H5T_CSET_ASCII;
115                      CTYPE H5T_C_S1;
116                   }
117                DATASPACE  SCALAR
118                DATA {
119                (0): "/entry/instrument/detector/two_theta"
120                }
121             }
122          }
123       }
124       GROUP "instrument" {
125          ATTRIBUTE "NX_class" {
126             DATATYPE  H5T_STRING {
127                   STRSIZE 12;
128                   STRPAD H5T_STR_NULLPAD;
129                   CSET H5T_CSET_ASCII;
130                   CTYPE H5T_C_S1;
131                }
132             DATASPACE  SCALAR
133             DATA {
134             (0): "NXinstrument"
135             }
136          }
137          GROUP "detector" {
138             ATTRIBUTE "NX_class" {
139                DATATYPE  H5T_STRING {
140                      STRSIZE 10;
141                      STRPAD H5T_STR_NULLPAD;
142                      CSET H5T_CSET_ASCII;
143                      CTYPE H5T_C_S1;
144                   }
145                DATASPACE  SCALAR
146                DATA {
147                (0): "NXdetector"
148                }
149             }
150             DATASET "counts" {
151                HARDLINK "/entry/data/counts"
152             }
153             DATASET "two_theta" {
154                HARDLINK "/entry/data/two_theta"
155             }
156          }
157       }
158    }
159 }
160 }
```

Look carefully! It *appears* from the output of `h5dump` that the actual data for `two_theta` and `counts` has *moved* into the `NXdata` group at HDF5 path `/entry/data`! But we stored that data in the `NXdetector` group at `/entry/instrument/detector`. This is normal for `h5dump` output.

A bit of explanation is necessary at this point. The data is not stored in either HDF5 group directly. Instead, HDF5

creates a DATA storage element in the file and posts a reference to that DATA storage element as needed. An HDF5 *hard link* requests another reference to that same DATA storage element. The h5dump tool describes in full that DATA storage element the first time (alphabetically) it is called. In our case, that is within the NXdata group. The next time it is called, within the NXdetector group, h5dump reports that a hard link has been made and shows the HDF5 path to the description.

NeXus recognizes this behavior of the HDF5 library and adds an additional structure when building hard links, the target attribute, to preserve the original location of the data. Not that it actually matters. The h5toText.py tool knows about the additional NeXus target attribute and shows the data to appear in its original location, in the NXdetector group.

```
writer_2_1.hdf5:NeXus data file
  entry:NXentry
    @NX_class = NXentry
    data:NXdata
      @NX_class = NXdata
      counts --> /entry/instrument/detector/counts
      two_theta --> /entry/instrument/detector/two_theta
    instrument:NXinstrument
      @NX_class = NXinstrument
      detector:NXdetector
        @NX_class = NXdetector
        counts:NX_INT32[31] = __array
          @units = counts
          @signal = 1
          @axes = two_theta
          @target = /entry/instrument/detector/counts
          __array = [1037, 1318, 1704, '...', 1321]
        two_theta:NX_FLOAT64[31] = __array
          @units = degrees
          @target = /entry/instrument/detector/two_theta
          __array = [17.926079999999999, 17.925909999999998,
                     17.925750000000001, '...', 17.92108]
```

Complete h5py example writing and reading a NeXus data file

Writing the HDF5 file using h5py In the main code section of *BasicWriter.py*, a current time stamp is written in the format of *ISO 8601*. For simplicity of this code example, we use a text string for the time, rather than computing it directly from Python support library calls. It is easier this way to see the exact type of string formatting for the time. When using the Python datatime package, one way to write the time stamp is:

```
timestamp = "T".join( str( datetime.datetime.now() ).split() )
```

The data (mr is similar to "two_theta" and I00 is similar to "counts") is collated into two Python lists. We use our my_lib support to read the file and parse the two-column format.

The new HDF5 file is opened (and created if not already existing) for writing, setting common NeXus attributes in the same command from our support library. Proper HDF5+NeXus groups are created for /entry:NXentry/mr_scan:NXdata. Since we are not using the NAPI, our support library must create and set the NX_class attribute on each group.

Note: We want to create the desired structure of /entry:NXentry/mr_scan:NXdata/. First, our support library calls nxentry = f.create_group("entry") to create the NXentry group called entry at the root level. Then, it calls nxdata = nxentry.create_group("mr_scan") to create the NXentry group called entry as a child of the NXentry group.

Next, we create a dataset called `title` to hold a title string that can appear on the default plot.

Next, we create datasets for `mr` and `I00` using our support library. The data type of each, as represented in `numpy`, will be recognized by `h5py` and automatically converted to the proper HDF5 type in the file. A Python dictionary of attributes is given, specifying the engineering units and other values needed by NeXus to provide a default plot of this data. By setting `signal=1` as an attribute on `I00`, NeXus recognizes `I00` as the default *y* axis for the plot. The `axes="mr"` connects the dataset to be used as the *x* axis.

Finally, we *must* remember to call `f.close()` or we might corrupt the file when the program quits.

BasicWriter.py: Write a NeXus HDF5 file using Python with h5py

```
#!/usr/bin/env python
'''Writes a NeXus HDF5 file using h5py and numpy'''

import h5py    # HDF5 support
import numpy
import my_lib  # uses h5py

print "Write a NeXus HDF5 file"
fileName = "prj_test.nexus.hdf5"
timestamp = "2010-10-18T17:17:04-0500"

# load data from two column format
data = numpy.loadtxt('input.dat').T
mr_arr = data[0]
i00_arr = numpy.asarray(data[1],'int32')

# create the HDF5 NeXus file
f = my_lib.makeFile(fileName, file_name=fileName,
        file_time=timestamp,
        instrument="APS USAXS at 32ID-B",
        creator="$Id: BasicWriter.py 1194 2013-01-13 22:17:36Z Pete Jemian $",
        NeXus_version="4.3.0",
        HDF5_Version=h5py.version.hdf5_version,
        h5py_version=h5py.version.version)

nxentry = my_lib.makeGroup(f, "entry", "NXentry")
my_lib.makeDataset(nxentry, 'title', data='1-D scan of I00 v. mr')

nxdata = my_lib.makeGroup(nxentry, "mr_scan", "NXdata")

my_lib.makeDataset(nxdata,  "mr",  mr_arr, units='degrees',
                   long_name='USAXS mr (degrees)')

my_lib.makeDataset(nxdata,  "I00",  i00_arr, units='counts',
      signal=1,          # Y axis of default plot
      axes='mr',         # name "mr" as X axis
      long_name='USAXS I00 (counts)')

f.close()    # be CERTAIN to close the file

print "wrote file:", fileName
```

Reading the HDF5 file using h5py The file reader, *BasicReader.py*, is very simple since the bulk of the work is done by `h5py`. Our code opens the HDF5 we wrote above, prints the HDF5 attributes from the file, reads the two

datasets, and then prints them out as columns. As simple as that. Of course, real code might add some error-handling and extracting other useful stuff from the file.

Note: See that we identified each of the two datasets using HDF5 absolute path references (just using the group and dataset names). Also, while coding this example, we were reminded that HDF5 is sensitive to upper or lowercase. That is, `I00` is not the same is `i00`.

BasicReader.py: Read a NeXus HDF5 file using Python with h5py

```
#!/usr/bin/env python
'''Reads NeXus HDF5 files using h5py and prints the contents'''

import h5py    # HDF5 support

fileName = "prj_test.nexus.hdf5"
f = h5py.File(fileName,  "r")
for item in f.attrs.keys():
    print item + ":", f.attrs[item]
mr = f['/entry/mr_scan/mr']
i00 = f['/entry/mr_scan/I00']
print "%s\t%s\t%s" % ("#", "mr", "I00")
for i in range(len(mr)):
    print "%d\t%g\t%d" % (i, mr[i], i00[i])
f.close()
```

Output from `BasicReader.py` is shown next.

Output from `BasicReader.py`

```
file_name: prj_test.nexus.hdf5
file_time: 2010-10-18T17:17:04-0500
creator: $Id: BasicWriter.py 647 2010-10-19 22:34:01Z Pete Jemian $
HDF5_Version: 1.8.5
NeXus_version: 4.3.0
h5py_version: 1.2.1
instrument: APS USAXS at 32ID-B
#   mr  I00
0   17.9261 1037
1   17.9259 1318
2   17.9258 1704
3   17.9256 2857
4   17.9254 4516
5   17.9252 9998
6   17.9251 23819
7   17.9249 31662
8   17.9247 40458
9   17.9246 49087
10  17.9244 56514
11  17.9243 63499
12  17.9241 66802
13  17.9239 66863
14  17.9237 66599
15  17.9236 66206
16  17.9234 65747
```

```
17  17.9232 65250
18  17.9231 64129
19  17.9229 63044
20  17.9228 60796
21  17.9226 56795
22  17.9224 51550
23  17.9222 43710
24  17.9221 29315
25  17.9219 19782
26  17.9217 12992
27  17.9216 6622
28  17.9214 4198
29  17.9213 2248
30  17.9211 1321
```

Validating the HDF5 file Now we have an HDF5 file that contains our data. What makes this different from a NeXus data file? A NeXus file has a specific arrangement of groups and datasets in an HDF5 file.

To test that our HDF5 file conforms to the NeXus standard, we use the *Java-version of NXvalidate*. Referring to the next figure, we compare our HDF5 file with the rules for generic [6] data files (`all.nxdl.xml`). The only items that have been flagged are the "non-standard field names" *mr* and *I00*. Neither of these two names is specifically named in the NeXus NXDL definition for the `NXdata` base class. As we'll see shortly, this is not a problem.

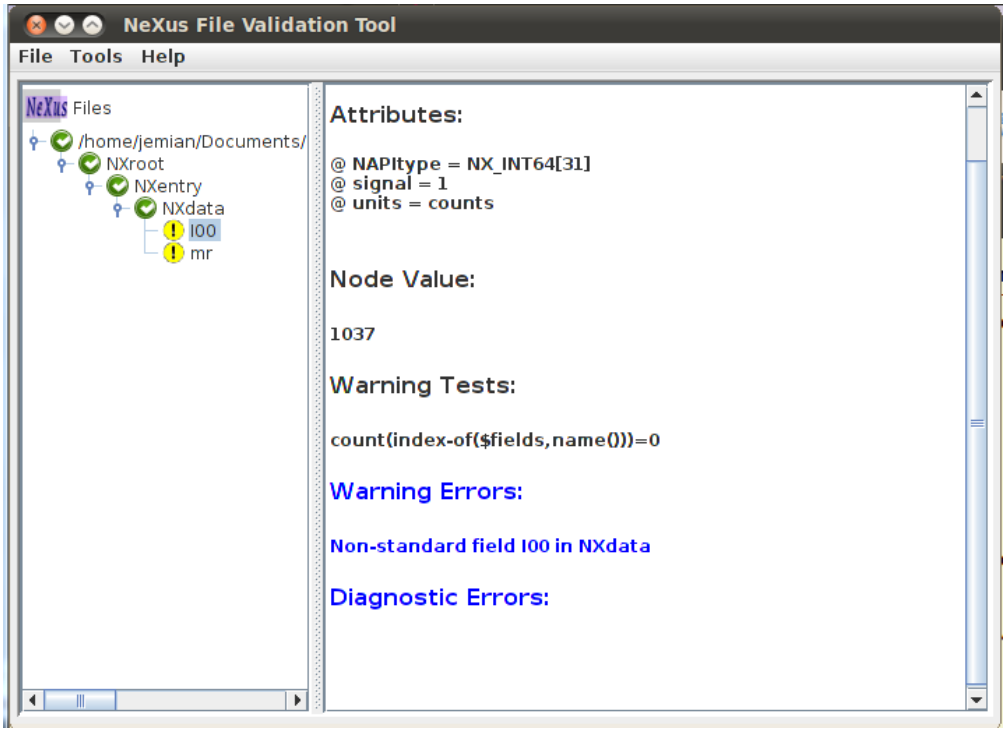


Figure 2.14: NeXus validation of our HDF5 file

Note: Note that `NXvalidate` shows only the first data field for *mr* and *I00*.

[6] generic NeXus data files: NeXus data files for which no application-specific NXDL applies

Plotting the HDF5 file Now that we are certain our file conforms to the NeXus standard, let's plot it using the `NeXpy` [7] client tool. To help label the plot, we added the `long_name` attributes to each of our datasets. We also added metadata to the root level of our HDF5 file similar to that written by the NAPI. It seemed to be a useful addition. Compare this with *plot of our mr_scan* and note that the horizontal axis of this plot is mirrored from that above. This is because the data is stored in the file in descending `mr` order and `NeXpy` has plotted it that way by default.

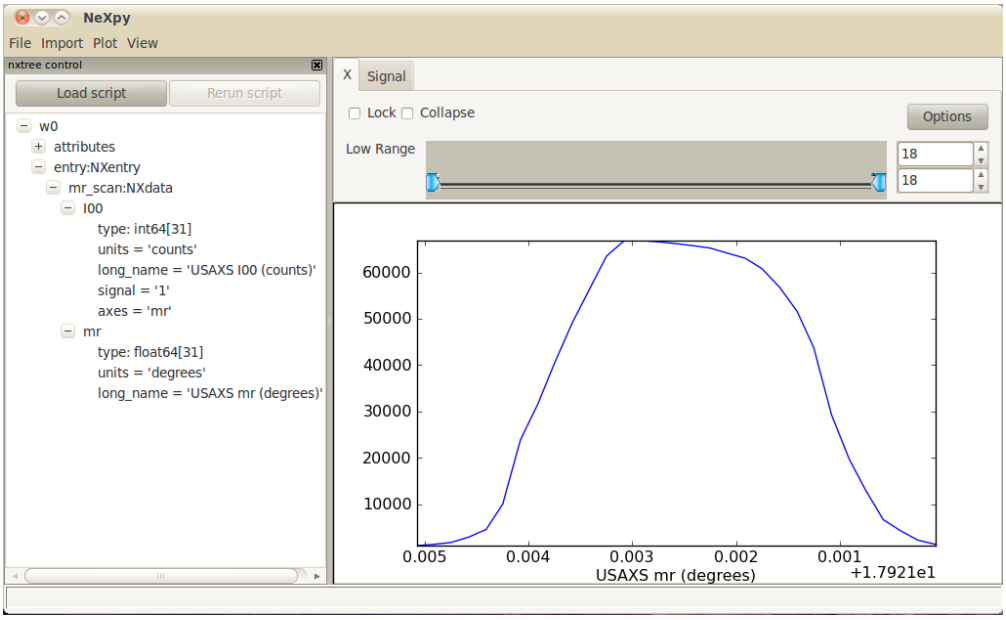


Figure 2.15: plot of our *mr_scan* using NeXpy

Links to Data in External HDF5 Files

HDF5 files may contain links to data (or groups) in other files. This can be used to advantage to refer to data in existing HDF5 files and create NeXus-compliant data files. Here, we show such an example, using the same `counts` v. `two_theta` data from the examples above.

file: external_angles.hdf5 Take for example, the structure of `external_angles.hdf5`, a simple HDF5 data file that contains just the `two_theta` angles in an HDF5 dataset at the root level of the file. Although this is a valid HDF5 data file, it is not a valid NeXus data file:

```
angles:float64[31] = [17.926079999999999, '...', 17.92108]
  @units = degrees
```

file: external_counts.hdf5 The data in the file `external_angles.hdf5` might be referenced from another HDF5 file (such as `external_counts.hdf5`) by an HDF5 external link. [8] Here is an example of the structure

```
entry:NXentry
  instrument:NXinstrument
    detector:NXdetector
      counts:NX_INT32[31] = [1037, '...', 1321]
```

[7] *NeXpy*: http://nexpy.github.io/nexpy/

[8] see these URLs for further guidance on HDF5 external links: http://www.hdfgroup.org/HDF5/doc/RM/RM_H5L.html#Link-CreateExternal, http://www.h5py.org/docs-1.3/guide/group.html#external-links

```
            @units = counts
            @signal = 1
            @axes = two_theta
          two_theta --> file="external_angles.hdf5", path="/angles"
```

Note: The file `external_counts.hdf5` is not a complete NeXus file since it does not contain an NXdata group containing a dataset with `signal=1` attribute.

---

file: external_master.hdf5 A valid NeXus data file could be created that refers to the data in these files without making a copy of the data files themselves.

Note: It is necessary for all these files to be located together in the same directory for the HDF5 external file links to work properly.‘

To be a valid NeXus file, it must contain a *NXentry* group containing a *NXdata* group containing only one dataset with the aatribute `signal=1`. For the files above, it is simple to make a master file that links to the data we desire, from structure that we create. In `external_counts.hdf5` above, see that the required attribute `signal=1` is already present. Here is `external_master.hdf5`, an example:

```
    entry:NXentry
      instrument --> file="external_counts.hdf5", path="/entry/instrument"
      data:NXdata
        counts --> file="external_counts.hdf5", path="/entry/instrument/detector/counts"
        two_theta --> file="external_angles.hdf5", path="/angles"
```

source code: externalExample.py Here is the complete code of a Python program, using `h5py` to write a NeXus-compliant HDF5 file with links to data in other HDF5 files.

***externalExample.py*: Write using HDF5 external links**

```
#!/usr/bin/env python
'''
Writes a NeXus HDF5 file using h5py with links to data in other HDF5 files.

This example is based on ''writer_2_1''.
'''

import my_lib

FILE_INPUT = 'input.dat'
FILE_HDF5_MASTER = 'external_master.hdf5'
FILE_HDF5_ANGLES = 'external_angles.hdf5'
FILE_HDF5_COUNTS = 'external_counts.hdf5'

#---------------------------

# get some data
tthData, countsData = my_lib.get2ColumnData(FILE_INPUT)

# put the angle data in an external (non-NeXus) HDF5 data file
f = my_lib.makeFile(FILE_HDF5_ANGLES)  # create an HDF5 file (non-NeXus)
tth = my_lib.makeDataset(f, "angles", tthData, units='degrees')
```

```
f.close()    # be CERTAIN to close the file


# put the detector counts in an external NeXus HDF5 data file
f = my_lib.makeFile(FILE_HDF5_COUNTS)
nxentry = my_lib.makeGroup(f, 'entry', 'NXentry')
nxinstrument = my_lib.makeGroup(nxentry, 'instrument', 'NXinstrument')
nxdetector = my_lib.makeGroup(nxinstrument, 'detector', 'NXdetector')
counts = my_lib.makeDataset(nxdetector, "counts", countsData,
                  units='counts', signal=1, axes='two_theta')
# make a link since "two_theta" has not been stored here
my_lib.makeExternalLink(f, FILE_HDF5_ANGLES,
                        '/angles', nxdetector.name+'/two_theta')
f.close()

# create a master NeXus HDF5 file
f = my_lib.makeFile(FILE_HDF5_MASTER)
nxentry = my_lib.makeGroup(f, 'entry', 'NXentry')
nxdata = my_lib.makeGroup(nxentry, 'data', 'NXdata')
my_lib.makeExternalLink(f, FILE_HDF5_ANGLES,
                        '/angles', nxdata.name+'/two_theta')
my_lib.makeExternalLink(f, FILE_HDF5_COUNTS,
                        '/entry/instrument/detector/counts',
                        nxdata.name+'/counts')
my_lib.makeExternalLink(f, FILE_HDF5_COUNTS,
                        '/entry/instrument',
                        nxentry.name+'/instrument')
f.close()
```

Python Helper Modules for `h5py` Examples

Two additional Python modules were used to describe these `h5py` examples. The source code for each is given here. The first is a library we wrote that helps us create standard NeXus components using `h5py`. The second is a tool that helps us inspect the content and structure of HDF5 files.

`mylib` support module The examples in this section make use of a small helper library that calls `h5py` to create the various NeXus data components of *Data Groups*, *Data Fields*, *Data Attributes*, and *Links*. In a smaller sense, this subroutine library (`my_lib`) fills the role of the NAPI for writing the data using h5py.

```
#!/usr/bin/env python
'''
my_lib: routines to support reading & writing NeXus HDF5 files using h5py
'''

import h5py    # HDF5 support
import numpy   # in this case, provides data structures

def makeFile(filename, **attr):
    """
    create and open an empty NeXus HDF5 file using h5py

    Any named parameters in the call to this method will be saved as
    attributes of the root of the file.
    Note that **attr is a dictionary of named parameters.

    :param str filename: valid file name
```

```
    :param attr: optional keywords of attributes
    :return: h5py file object
    """
    obj = h5py.File(filename, "w")
    addAttributes(obj, attr)
    return obj

def makeGroup(parent, name, nxclass, **attr):
    """
    create a NeXus group

    Any named parameters in the call to this method
    will be saved as attributes of the group.
    Note that **attr is a dictionary of named parameters.

    :param obj parent: parent group
    :param str name: valid NeXus group name
    :param str nxclass: valid NeXus class name
    :param attr: optional keywords of attributes
    :return: h5py group object
    """
    obj = parent.create_group(name)
    obj.attrs["NX_class"] = nxclass
    addAttributes(obj, attr)
    return obj

def makeDataset(parent, name, data = None, **attr):
    '''
    create and write data to a dataset in the HDF5 file hierarchy

    Any named parameters in the call to this method
    will be saved as attributes of the dataset.

    :param obj parent: parent group
    :param str name: valid NeXus dataset name
    :param obj data: the data to be saved
    :param attr: optional keywords of attributes
    :return: h5py dataset object
    '''
    if data == None:
        obj = parent.create_dataset(name)
    else:
        obj = parent.create_dataset(name, data=data)
    addAttributes(obj, attr)
    return obj

def makeLink(parent, sourceObject, targetName):
    """
    create an internal NeXus (hard) link in an HDF5 file

    :param obj parent: parent group of source
    :param obj sourceObject: existing HDF5 object
    :param str targetName: HDF5 node path to be created,
                           such as ''/entry/data/data''
    """
    if not 'target' in sourceObject.attrs:
        # NeXus link, NOT an HDF5 link!
        sourceObject.attrs["target"] = str(sourceObject.name)
```

```
    parent._id.link(sourceObject.name, targetName, h5py.h5g.LINK_HARD)

def makeExternalLink(hdf5FileObject, sourceFile, sourcePath, targetPath):
    """
    create an external link from sourceFile, sourcePath to targetPath in hdf5FileObject

    :param obj hdf5FileObject: open HDF5 file object
    :param str sourceFile: file containing existing HDF5 object at sourcePath
    :param str sourcePath: path to existing HDF5 object in sourceFile
    :param str targetPath: full node path to be created in current open HDF5 file,
                           such as ``/entry/data/data``

    .. note::
       Since the object retrieved is in a different file,
       its ".file" and ".parent" properties will refer to
       objects in that file, not the file in which the link resides.

    .. see:: http://www.h5py.org/docs-1.3/guide/group.html#external-links

    This routine is provided as a reminder how to do this simple operation.
    """
    hdf5FileObject[targetPath] = h5py.ExternalLink(sourceFile, sourcePath)

def addAttributes(parent, attr):
    """
    add attributes to an h5py data item

    :param obj parent: h5py parent object
    :param dict attr: dictionary of attributes
    """
    if attr and type(attr) == type({}):
        # attr is a dictionary of attributes
        for k, v in attr.items():
            parent.attrs[k] = v

def get2ColumnData(fileName):
    '''
    read two-column data from a file,
    first column is float,
    second column is integer
    '''
    buffer = numpy.loadtxt(fileName).T
    xArr = buffer[0]
    yArr = numpy.asarray(buffer[1],'int32')
    return xArr, yArr
```

`h5toText` support module The module `h5toText` reads an HDF5 data file and prints out the structure of the groups, datasets, attributes, and links in that file. There is a command-line option to print out more or less of the data in the dataset arrays.

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

'''
Print the structure of an HDF5 file to stdout

$Id: h5toText.py 1194 2013-01-13 22:17:36Z Pete Jemian $
'''
```

```
8
9
10  ############ SVN repository information ###################
11  # $Date: 2013-01-13 22:17:36 +0000 (Sun, 13 Jan 2013) $
12  # $Author: Pete Jemian $
13  # $Revision: 1194 $
14  # $URL: file:///isis/svn/nexus/definitions/trunk/manual/source/examples/h5py/h5toText.py $
15  # $Id: h5toText.py 1194 2013-01-13 22:17:36Z Pete Jemian $
16  ############ SVN repository information ###################
17
18
19  import h5py
20  import os
21  import sys
22  import getopt
23
24
25  class H5toText(object):
26      '''
27      Example usage showing default display::
28
29          mc = H5toText(filename)
30          mc.array_items_shown = 5
31          mc.report()
32      '''
33      filename = None
34      requested_filename = None
35      isNeXus = False
36      array_items_shown = 5
37
38      def __init__(self, filename, makeReport = False):
39          ''' Constructor '''
40          self.requested_filename = filename
41          if os.path.exists(filename):
42              self.filename = filename
43              self.isNeXus = self.testIsNeXus()
44              if makeReport:
45                  self.report()
46
47      def report(self):
48          ''' reporter '''
49          if self.filename == None: return
50          f = h5py.File(self.filename, 'r')
51          txt = self.filename
52          if self.isNeXus:
53              txt += ":NeXus data file"
54          self.showGroup(f, txt, indentation = "")
55          f.close()
56
57      def testIsNeXus(self):
58          '''
59          test if the selected HDF5 file is a NeXus file
60
61          At this time, the code only tests for the existence of
62          the NXentry group.  The tests should be extended to require
63          a NXdata group and a single dataset containing signal=1 attribute.
64          '''
65          result = False
```

```
        try:
            f = h5py.File(self.filename, 'r')
            for value in f.itervalues():
                #print str(type(value))
                if '.Group' not in str(type(value)):
                    continue
                #print value.attrs.keys()
                if 'NX_class' not in value.attrs:
                    continue
                v = value.attrs['NX_class']
                #print type(v), v, type("a string")
                possible_types = ["<type 'numpy.string_'>", ]
                possible_types.append("<type 'str'>")
                if str(type(v)) not in possible_types:
                    continue
                if str(v) == str('NXentry'):
                    # TODO: apply more tests
                    #    for group NXdata
                    #    and signal=1 attribute on only one dataset
                    result = True
                    break
            f.close()
        except:
            pass
        return result

    def showGroup(self, obj, name, indentation = "  "):
        '''print the contents of the group'''
        nxclass = ""
        if 'NX_class' in obj.attrs:
            class_attr = obj.attrs['NX_class']
            nxclass = ":" + str(class_attr)
        print indentation + name + nxclass
        self.showAttributes(obj, indentation)
        # show datasets and links next
        groups = []
        for itemname in sorted(obj):
            linkref = obj.get(itemname, getlink=True)
            if '.ExternalLink' in str(type(linkref)):
                # if the external file is not present, cannot know if
                # link target is a dataset or a group or another link
                fmt = '%s  %s --> file="%s", path="%s"'
                print fmt % (indentation, itemname, linkref.filename, linkref.path)
            else:
                classref = obj.get(itemname, getclass=True)
                value = obj.get(itemname)
                if '.File' in str(classref) or '.Group' in str(classref):
                    groups.append(value)
                elif '.Dataset' in str(classref):
                    self.showDataset(value, itemname, indentation+"  ")
                else:
                    msg = "unidentified %s: %s, %s", itemname, repr(classref), repr(linkref)
                    raise Exception, msg
        # then show things that look like groups
        for value in groups:
            itemname = value.name.split("/")[-1]
            self.showGroup(value, itemname, indentation+"  ")

```

```
124     def showAttributes(self, obj, indentation = "  "):
125         '''print any attributes'''
126         for name, value in obj.attrs.iteritems():
127             print "%s  @%s = %s" % (indentation, name, str(value))
128 
129     def showDataset(self, dset, name, indentation = "  "):
130         '''print the contents and structure of a dataset'''
131         shape = dset.shape
132         if self.isNeXus:
133             if "target" in dset.attrs:
134                 if dset.attrs['target'] != dset.name:
135                     print "%s%s --> %s" % (indentation, name,
136                                            dset.attrs['target'])
137                     return
138         txType = self.getType(dset)
139         txShape = self.getShape(dset)
140         if shape == (1,):
141             value = " = %s" % str(dset[0])
142             print "%s%s:%s%s%s" % (indentation, name, txType,
143                                    txShape, value)
144             self.showAttributes(dset, indentation)
145         else:
146         print "%s%s:%s%s = __array" % (indentation, name,
147                                        txType, txShape)
148             # show these before __array
149             self.showAttributes(dset, indentation)
150             if self.array_items_shown > 2:
151                 value = self.formatArray(dset, indentation + '  ')
152                 print "%s  %s = %s" % (indentation, "__array", value)
153             else:
154                 print "%s  %s: %s" % (indentation, "__array", "not shown")
155 
156     def getType(self, obj):
157         ''' get the storage (data) type of the dataset '''
158         t = str(obj.dtype)
159         if t[0:2] == '|S':
160             t = 'char[%s]' % t[2:]
161         if self.isNeXus:
162             t = 'NX_' + t.upper()
163         return t
164 
165     def getShape(self, obj):
166         ''' return the shape of the HDF5 dataset '''
167         s = obj.shape
168         l = []
169         for dim in s:
170             l.append(str(dim))
171         if l == ['1']:
172             result = ""
173         else:
174             result = "[%s]" % ",".join(l)
175         return result
176 
177     def formatArray(self, obj, indentation = '  '):
178         ''' nicely format an array up to rank=5 '''
179         shape = obj.shape
180         r = ""
181         if len(shape) in (1, 2, 3, 4, 5):
```

```
182             r = self.formatNdArray(obj, indentation + '  ')
183         if len(shape) > 5:
184             r = "### no arrays for rank > 5 ###"
185         return r
186 
187     def decideNumShown(self, n):
188         ''' determine how many values to show '''
189         if self.array_items_shown != None:
190             if n > self.array_items_shown:
191                 n = self.array_items_shown - 2
192         return n
193 
194     def formatNdArray(self, obj, indentation = '  '):
195         ''' return a list of lower-dimension arrays, nicely formatted '''
196         shape = obj.shape
197         rank = len(shape)
198         if not rank in (1, 2, 3, 4, 5): return None
199         n = self.decideNumShown( shape[0] )
200         r = []
201         for i in range(n):
202             if rank == 1: item = obj[i]
203             if rank == 2: item = self.formatNdArray(obj[i, :])
204             if rank == 3: item = self.formatNdArray(obj[i, :, :],
205                                                     indentation + '  ')
206             if rank == 4: item = self.formatNdArray(obj[i, :, :, :],
207                                                     indentation + '  ')
208             if rank == 5: item = self.formatNdArray(obj[i, :, :, :, :],
209                                                     indentation + '  ')
210             r.append( item )
211         if n < shape[0]:
212             # skip over most
213             r.append("...")
214             #  get the last one
215             if rank == 1: item = obj[-1]
216             if rank == 2: item = self.formatNdArray(obj[-1, :])
217             if rank == 3: item = self.formatNdArray(obj[-1, :, :],
218                                                     indentation + '  ')
219             if rank == 4: item = self.formatNdArray(obj[-1, :, :, :],
220                                                     indentation + '  ')
221             if rank == 5: item = self.formatNdArray(obj[-1, :, :, :, :],
222                                                     indentation + '  ')
223             r.append( item )
224         if rank == 1:
225             s = str( r )
226         else:
227             s = "[\n" + indentation + '  '
228             s += ("\n" + indentation + '  ').join(r)
229             s += "\n" + indentation + "]"
230         return s
231 
232 
233 def do_filelist(filelist, limit=5):
234     '''
235     interpret the structure of a list of HDF5 files
236 
237     :param [str] filelist: one or more file names to be interpreted
238     :param int limit: maximum number of array items to be shown (default = 5)
239     '''
```

```
    for item in filelist:
        mc = H5toText(item)
        mc.array_items_shown = limit
        mc.report()


def do_test():
    limit = 3
    filelist = []
    filelist.append('th02c_ps02_1_master.h5')
    filelist.append('external_angles.hdf5')
    filelist.append('external_counts.hdf5')
    filelist.append('external_master.hdf5')
    filelist.append('../Create/example1.hdf5')
    filelist.append('../Create/example2.hdf5')
    filelist.append('../Create/example3.hdf5')
    filelist.append('../Create/example4.hdf5')
    filelist.append('../../../NeXus/definitions/trunk/manual/examples/h5py/prj_test.nexus.hdf5')
    filelist.append('../../../NeXus/definitions/exampledata/code/hdf5/dmc01.h5')
    filelist.append('../../../NeXus/definitions/exampledata/code/hdf5/dmc02.h5')
    filelist.append('../../../NeXus/definitions/exampledata/code/hdf5/focus2007n001335.hdf')
    filelist.append('../../../NeXus/definitions/exampledata/code/hdf5/NXtest.h5')
    filelist.append('../../../NeXus/definitions/exampledata/code/hdf5/sans2009n012333.hdf')
    filelist.append('../Create/simple5.nxs')
    filelist.append('../Create/bad.h5')

    do_filelist(filelist, limit)


def main():
    '''standard command-line interface'''
    try:
        opts, args = getopt.getopt(sys.argv[1:], "n:")
    except:
        print
        print "SVN: $Id: h5toText.py 1194 2013-01-13 22:17:36Z Pete Jemian $"
        print "usage: ", sys.argv[0], " [-n ##] HDF5_file_name [another_HDF5_file_name]"
        print "  -n ## : limit number of displayed array items to ## (must be 3 or more or 'None')"
        print
    for item in opts:
        if item[0] == "-n":
            if item[1].lower() == "none":
                limit = None
            else:
                limit = int(item[1])
    do_filelist(args)


if __name__ == '__main__':
    if len(sys.argv) > 1:
        main()
    else:
        do_test()
```

Viewing 2-D Data from LRMECS

The IPNS LRMECS instrument stored data in NeXus HDF4 data files. One such example is available from the repository of NeXus data file examples. For this example, we will start with a conversion of that original data file into *HDF5* format.

HDF4 http://svn.nexusformat.org/definitions/exampledata/IPNS/LRMECS/lrcs3701.nxs

HDF5 http://svn.nexusformat.org/definitions/exampledata/IPNS/LRMECS/lrcs3701.nx5

This dataset contains two histograms with 2-D images (148x750 and 148x32) of 32-bit integers. First, we use the `h5dump` tool to investigate the header content of the file (not showing any of the data).

Visualize Using `h5dump`

Here, the output of the command:

```
h5dump -H lrcs3701.nx5
```

has been edited to only show the first *NXdata* group (`/Histogram1/data`):

LRMECS `lrcs3701` data: `h5dump` output

```
HDF5 "C:\Users\Pete\Documents\eclipse\NeXus\definitions\exampledata\IPNS\LRMECS\lrcs3701.nx5" {
GROUP "/Histogram1/data" {
   DATASET "data" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 148, 750 ) / ( 148, 750 ) }
   }
   DATASET "polar_angle" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 148 ) / ( 148 ) }
   }
   DATASET "time_of_flight" {
      DATATYPE  H5T_IEEE_F32LE
      DATASPACE  SIMPLE { ( 751 ) / ( 751 ) }
   }
   DATASET "title" {
      DATATYPE  H5T_STRING {
            STRSIZE 44;
            STRPAD H5T_STR_NULLTERM;
            CSET H5T_CSET_ASCII;
            CTYPE H5T_C_S1;
         }
      DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
   }
}
}
```

Visualize Using *HDFview*

For many, the simplest way to view the data content of an HDF5 file is to use the *HDFview* program (http://www.hdfgroup.org/hdf-java-html/hdfview) from The HDF Group. After starting *HDFview*, the data file may be loaded by dragging it into the main HDF window. On opening up to the first `NXdata` group */Histogram1/data* (as above), and then double-clicking the dataset called: *data*, we get our first view of the data.

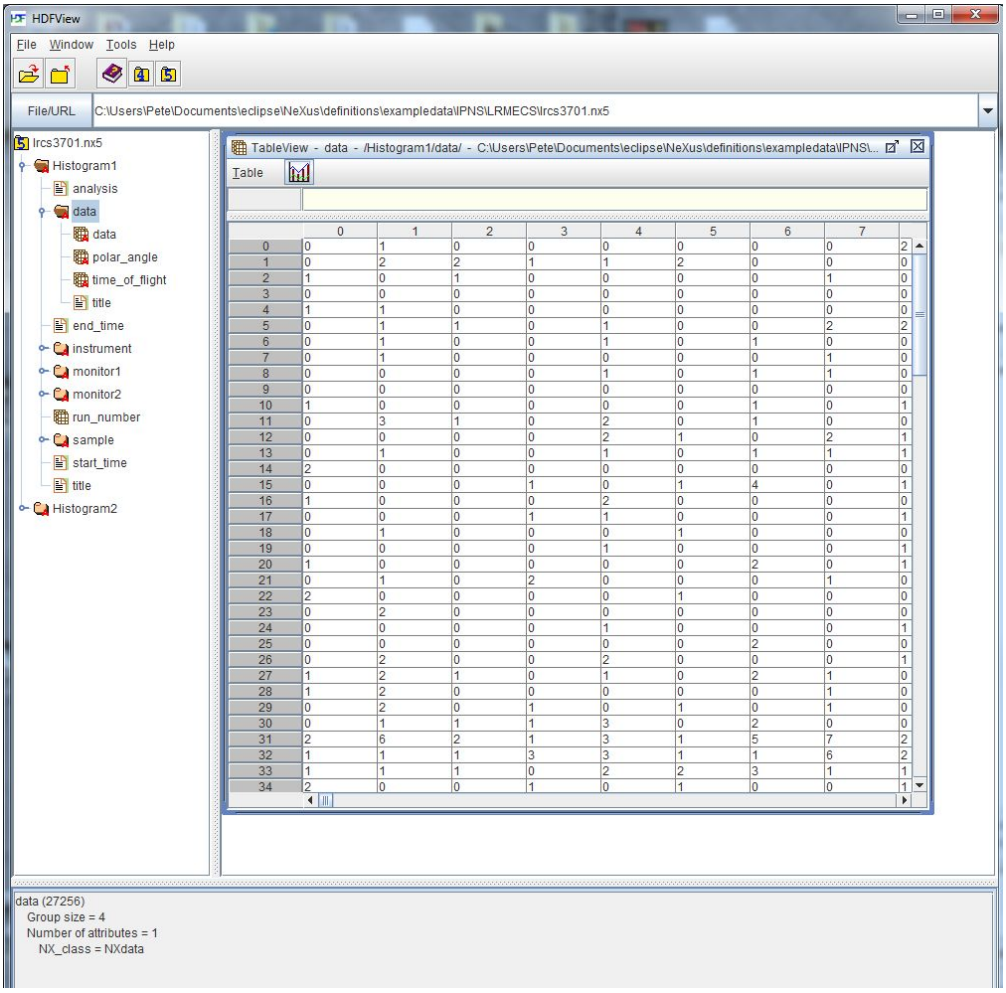


Figure 2.16: LRMECS `lrcs3701` data: *HDFview*

The data may be represented as an image by accessing the *Open As* menu from HDFview (on Windows, right click the dataset called *data* and select the *Open As* item, consult the HDFview documentation for different platform instructions). Be sure to select the *Image* radio button, and then (accepting everything else as a default) press the *Ok* button.

Note: In this image, dark represents low intensity while white represents high intensity.

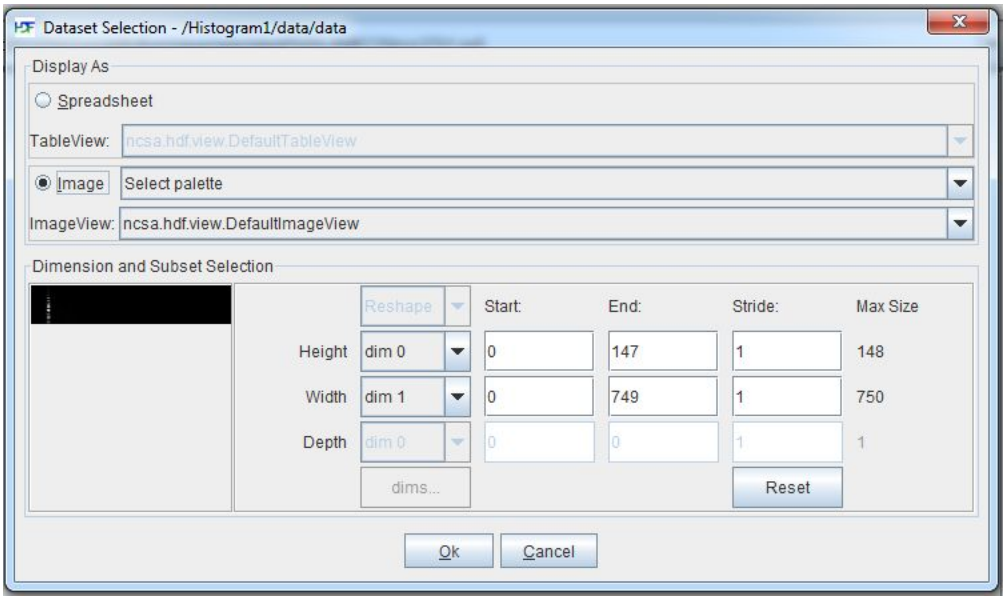


Figure 2.17: LRMECS `lrcs3701` data: *HDFview Open As* dialog

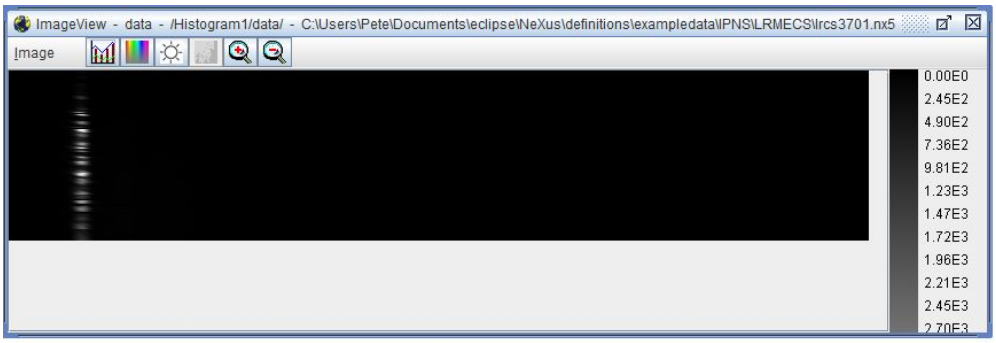


Figure 2.18: LRMECS `lrcs3701` data: *HDFview* Image

LRMECS `lrcs3701` data: image

Visualize Using *IgorPro*

Another way to visualize this data is to use a commercial package for scientific data visualization and analysis. One such package is *IgorPro* from http://www.wavemetrics.com

IgorPro provides a browser for HDF5 files that can open our NeXus HDF5 and display the image. Follow the instructions from WaveMetrics to install the *HDF5 Browser* package:

http://www.wavemetrics.com/products/igorpro/dataaccess/hdf5.htm

You may not have to do this step if you have already installed the *HDF5 Browser*. IgorPro will tell you if it is not installed properly. To install the *HDF5 Browser*, first start *IgorPro*. Next, select from the menus and submenus: `Data; Load Waves; Packages; Install HDF5 Package` as shown in the next figure. IgorPro may direct you to perform more activities before you progress from this step.

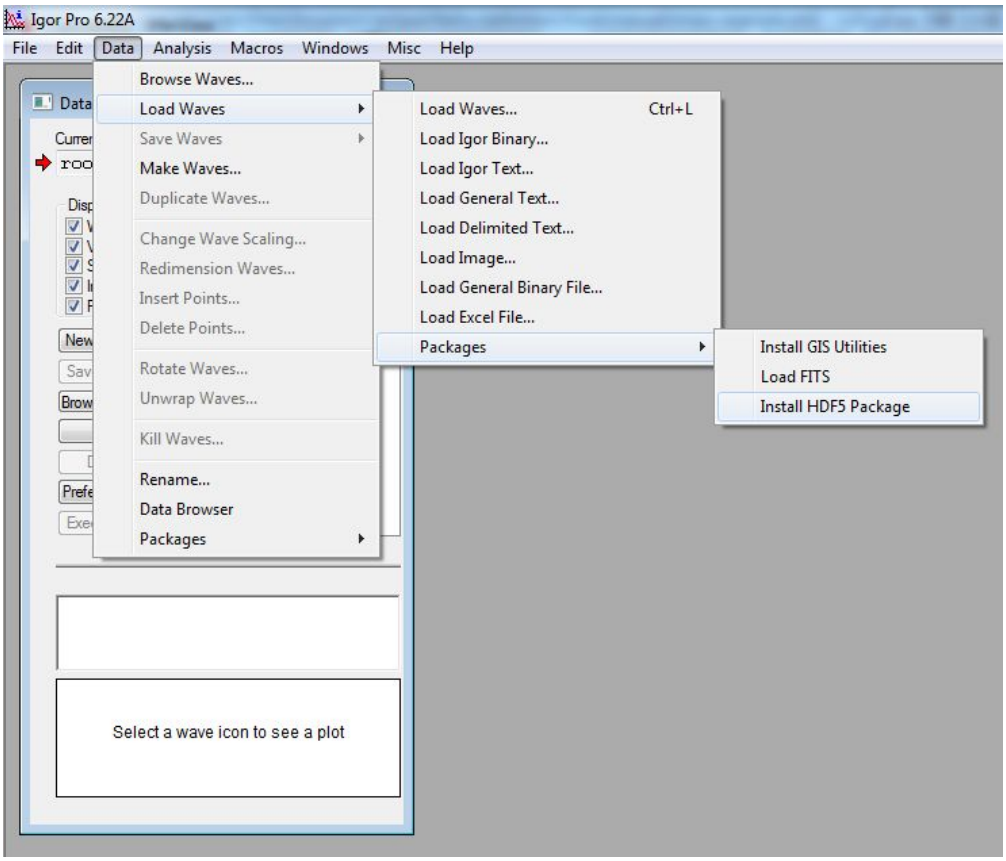


Figure 2.19: LRMECS `lrcs3701` data: *IgorPro* install HDF5 Browser

Next, open the *HDF5 Browser* by selecting from the menus and submenus: `Data; Load Waves; New HDF5 Browser` as shown in the next figure.

Next, click the *Open HDF5 File* button and open the NeXus HDF5 file `lrcs3701.nxs`. In the lower left *Groups* panel, click the *data* dataset. Also, under the panel on the right called *Load Dataset Options*, choose `No Table` as shown. Finally, click the *Load Dataset* button (in the *Datasets* group) to display the image.

Note: In this image, dark represents low intensity while white represents high intensity. The image has been rotated for easier representation in this manual.

LRMECS `lrcs3701` data: image

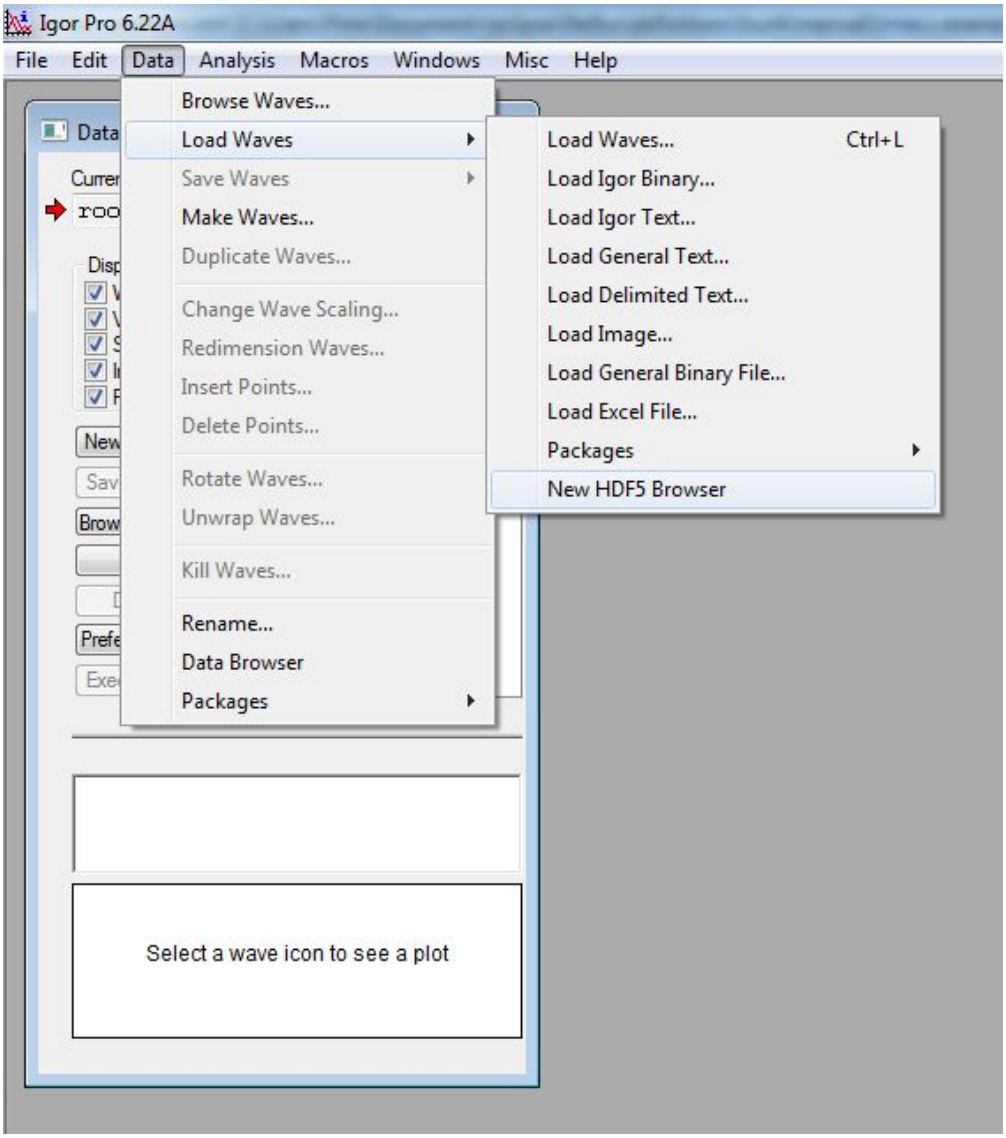


Figure 2.20: LRMECS `lrcs3701` data: *IgorPro HDFBrowser* dialog

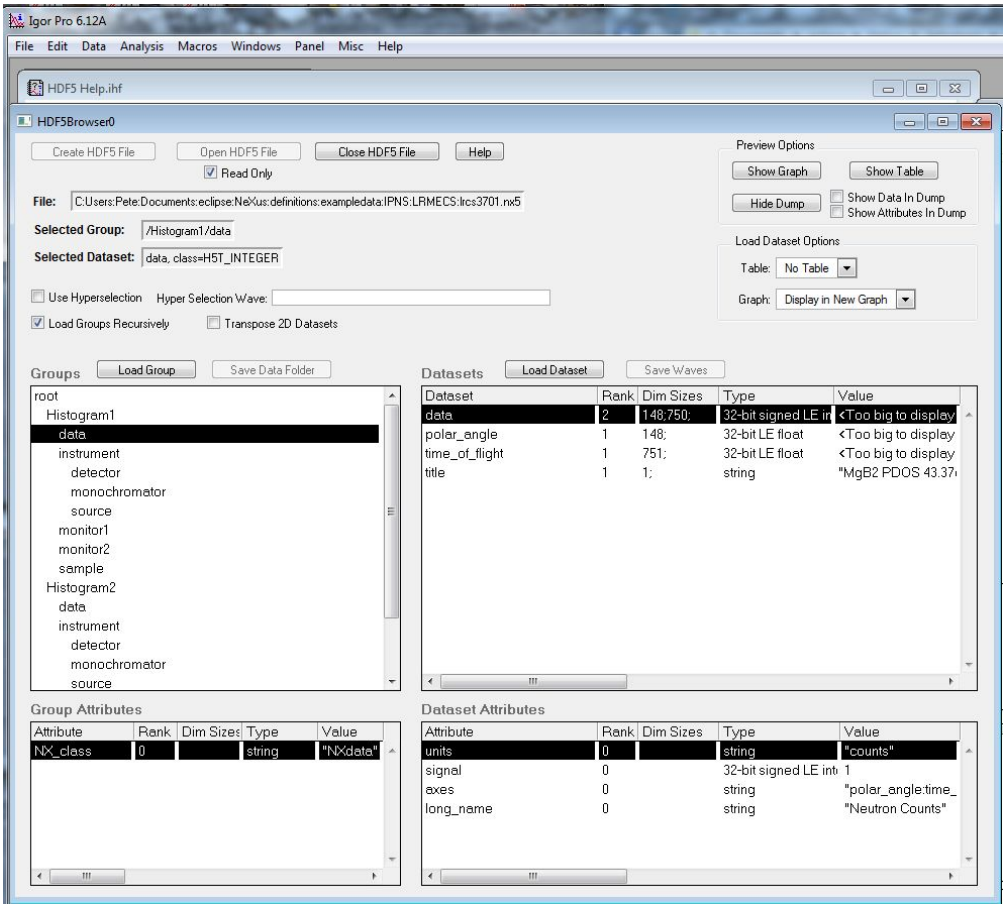


Figure 2.21: LRMECS `lrcs3701` data: *IgorPro HDFBrowser* dialog

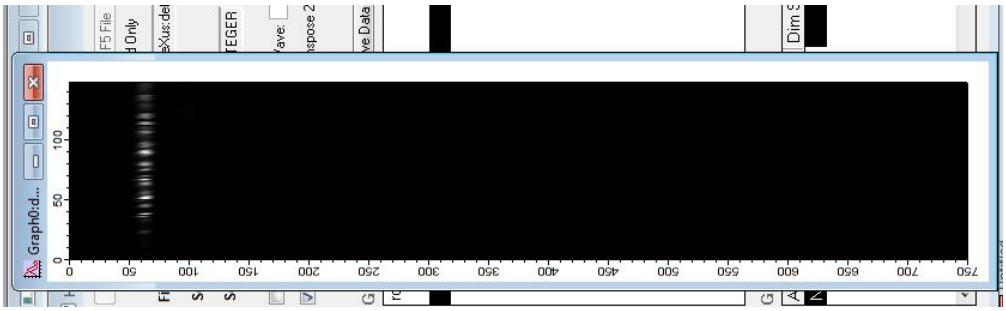


Figure 2.22: LRMECS `lrcs3701` data: *IgorPro* Image

CHAPTER

THREE

AUTHORS

Ray Osborn <rosborn@anl.gov>, Argonne National Laboratory, Argonne, IL, USA,

Mark Könnecke Mark Könnecke, <Mark.Koennecke@psi.ch>, Paul Scherrer Institut, CH-5232 Villigen PSI, Switzerland,

Przemek Klosowski <przemek.klosowski@nist.gov>, U. of Maryland and NIST, Gaithersburg, MD, USA,

Frederick Akeroyd <freddie.akeroyd@stfc.ac.uk>, Rutherford Appleton Laboratory, Didcot, UK,

Peter F. Peterson <petersonpf@ornl.gov>, Spallation Neutron Source, Oak Ridge, TN, USA,

Pete Jemian <jemian@anl.gov>, Advanced Photon Source, Argonne, IL, USA,

Stuart I. Campbell <campbellsi@ornl.gov>, Oak Ridge National Laboratory, Oak Ridge, TN, USA,

Tobias Richter <Tobias.Richter@diamond.ac.uk>, Diamond Light Source Ltd., Didcot, UK

CHAPTER

FOUR

NEXUS COMMUNITY

NeXus began as a group of scientists with the goal of defining a common data storage format to exchange experimental results and to exchange ideas about how to analyze them.

The NeXus Scientific Community provides the scientific data, advice, and continued involvement with the NeXus standard. NeXus provides a forum for the scientific community to exchange ideas in data storage through the NeXus wiki.

The NeXus International Advisory Committee (NIAC) supervises the development and maintenance of the NeXus common data format for neutron, X-ray, and muon science. The NIAC supervises a technical committee to oversee the NeXus Application Programmer Interface (NAPI) and the NeXus class definitions.

There are several mechanisms in place in order to coordinate the development of NeXus with the larger community.

4.1 NeXus Wiki

First of all, there is the NeXus wiki, http://wiki.nexusformat.org/, which provides all kinds of information, including membership, minutes, and discussions from the meetings of the NIAC and Technical Committee Code Camps, proposed designs for consideration by NeXus, the NeXus logo, as well as some legacy documentation that we have not quite managed to move into the manual.

4.2 Contributed Definitions

The community is encouraged to provide new definitions (*Base Class Definitions* or *Application Definitions*) for consideration in the NeXus standard. These community contributions will be entered in the *Contributed Definitions* and will be curated according to procedures set forth by the *NIAC: The NeXus International Advisory Committee*.

4.3 Other Ways NeXus Coordinates with the Scientific Community

4.3.1 NIAC: The NeXus International Advisory Committee

The purpose of the NeXus International Advisory Committee (NIAC) [1] is to supervise the development and maintenance of the NeXus common data format for neutron, X-ray, and muon science. This purpose includes, but is not limited to, the following activities.

[1] For more details about the NIAC constitution, procedures, and meetings, refer to the NIAC wiki page: http://wiki.nexusformat.org/NIAC The members of the NIAC may be reached by email: nexus-committee@nexusformat.org

1. To establish policies concerning the definition, use, and promotion of the NeXus format.
2. To ensure that the specification of the NeXus format is sufficiently complete and clear for its use in the exchange and archival of neutron, X-ray, and muon data.
3. To receive and examine all proposed amendments and extensions to the NeXus format. In particular, to ratify proposed instrument and group class definitions, to ensure that the data structures conform to the basic NeXus specification, and to ensure that the definitions of data items are clear and unambiguous and conform to accepted scientific usage.
4. To ensure that documentation of the NeXus format is sufficient, current, and available to potential users both on the internet and in other forms.
5. To coordinate with the developers of the NeXus Application Programming Interface to ensure that it supports the use of the NeXus format in the neutron, X-ray, and muon communities, and to promote other software development that will benefit users of the NeXus format.
6. To coordinate with other organizations that maintain and develop related data formats to ensure maximum compatibility.

The committee will meet at least once every other calendar year according to the following plan:

- In years coinciding with the NOBUGS series of conferences (once every two years), members of the entire NIAC will meet as a satellite meeting to NOBUGS, along with interested members of the community.
- In intervening years, the executive officers of the NIAC will attend, along with interested members of the NIAC. This is intended to be a working meeting with a small group.

4.3.2 NeXus Mailing Lists

There are several mailing lists associated with NeXus:

- *NeXus Mailing List*
- *NeXus International Advisory Committee (NIAC) Mailing List*
- *NeXus Developers Mailing List*
- *NeXus Code Subversion Mailing List*
- *NeXus Code Tickets Mailing List*
- *NeXus Definitions Subversion Mailing List*
- *NeXus Definitions Tickets Mailing List*

NeXus Mailing List

We invite anyone who is associated with neutron and/or X-ray synchrotron scattering and who wishes to be involved in the development and testing of the NeXus format to subscribe to this list. It is for the free discussion of all aspects of the design and operation of the NeXus format.

- List Address: nexus@nexusformat.org
- Subscriptions: http://lists.nexusformat.org/mailman/listinfo/nexus
- Archive: http://lists.nexusformat.org/pipermail/nexus

NeXus International Advisory Committee (NIAC) Mailing List

This list contains discussions of the *NIAC: The NeXus International Advisory Committee*, which oversees the development of the NeXus data format. Its members represent many of the major neutron and synchrotron scattering sources in the world. Membership and posting to this list are confined to the committee members, but the archives are public.

- List Address: nexus-committee@nexusformat.org
- Subscriptions: http://lists.nexusformat.org/mailman/listinfo/nexus-committee
- Archive: http://lists.nexusformat.org/pipermail/nexus-committee

NeXus Developers Mailing List

This mailing list was for discussions concerning the technical development of NeXus (the Definitions, NXDL, and the NeXus Application Program Interface). There was, however, much overlap with the general NeXus mailing list and so this separate list was closed in October 2012, but the archive of previous posting is still available.

- Archive: http://lists.nexusformat.org/pipermail/nexus-developers

NeXus Code Subversion Mailing List

Members of this list will receive an email whenever a commit is made to the *NeXus code repository*. This list cannot be posted to - all questions should instead be sent to the NeXus Developers Mailing List (nexus-developers@nexusformat.org).

- List Address: nexus-code-svn@nexusformat.org
- Subscriptions: http://lists.nexusformat.org/mailman/listinfo/nexus-code-svn
- Archive: http://lists.nexusformat.org/pipermail/nexus-code-svn

NeXus Code Tickets Mailing List

Members of this list will receive an email whenever a ticket (bug/issue/task) associated with NeXus code library development is modified on the Nexus *code* TRAC server. The list of ticket updates and subversion changesets is available on the *code* repository TRAC timeline. This list does not accept postings - see *Issue Reporting*.

- List Address: nexus-code-tickets@nexusformat.org
- Subscriptions: http://lists.nexusformat.org/mailman/listinfo/nexus-code-tickets
- Archive: http://lists.nexusformat.org/pipermail/nexus-code-tickets
- TRAC Timeline: http://trac.nexusformat.org/code/report/1

NeXus Definitions Subversion Mailing List

Members of this list will receive an email whenever a commit is made to the *NeXus definitions repository*. This list cannot be posted to - all questions should instead be sent to the NeXus Developers Mailing List (nexus-developers@nexusformat.org).

- List Address: nexus-definitions-svn@nexusformat.org
- Subscriptions: http://lists.nexusformat.org/mailman/listinfo/nexus-definitions-svn
- Archive: http://lists.nexusformat.org/pipermail/nexus-definitions-svn

NeXus Definitions Tickets Mailing List

Members of this list will receive an email whenever a ticket (bug/issue/task) associated with NeXus definitions development is modified on the NeXus *definitions* TRAC server. The list of ticket updates and subversion changesets is available on the *definitions* repository TRAC timeline. This list does not accept postings - see *Issue Reporting*.

- List Address: nexus-definitions-tickets@nexusformat.org
- Subscriptions: http://lists.nexusformat.org/mailman/listinfo/nexus-definitions-tickets
- Archive: http://lists.nexusformat.org/pipermail/nexus-definitions-tickets
- TRAC Timeline: http://trac.nexusformat.org/definitions/report/1

4.3.3 NeXus Subversion Repositories

NeXus NXDL class definitions (both base classes and instruments) and the NeXus code library source are held in a pair of *subversion* [2] repositories. To locate specific releases of the NeXus code and definitions repositories, please consult the section titled *Releases*.

The repositories are world readable. You can browse them directly:

NeXus code library and applications http://svn.nexusformat.org/code

NeXus NXDL class definitions http://svn.nexusformat.org/definitions

Better looking interfaces are also provided using *ViewVC* [3] and *TRAC* [4]. These software tools have been added to the NeXus WWW site server.

Browse the NeXus version control repositories:

- *code* repository (NAPI, library, and applications)
  - **ViewVC** http://svn.nexusformat.org/viewvc/NeXusCode
  - **TRAC** http://trac.nexusformat.org/code/browser
- *definitions* repository (NXDL classes and manual source)
  - **ViewVC** http://svn.nexusformat.org/viewvc/NeXusDefinitions
  - **TRAC** http://trac.nexusformat.org/definitions/browser

The repository can also be interrogated for recent updates via a *query form*, such as:

```
http://svn.nexusformat.org/viewvc/NeXusCode/trunk/?view=queryform
```

For example, show me all changes in the last month for the code (library and applications) repository:

```
http://svn.nexusformat.org/viewvc/NeXusCode/trunk/?view=query&date=month&limit_changes=100
```

or the from the **Definition repository**:

```
http://trac.nexusformat.org/definitions/timeline?daysback=30
```

If you wish to receive an email when a change is made to the repository you should join the appropriate *Mailing Lists*.

XML RSS Feed	icon
Alternatively, you can use an RSS feed to keep abreast of changes. TRAC provides a link to its RSS feed on pages with an orange *XML RSS Feed* icon at their foot such as:	

[2] **ViewVC**: http://www.viewvc.org
[3] **TRAC**: http://trac.edgewall.org
[4] *subversion*, revision control software: http://subversion.apache.org

There are pages that show the subversion repository activity in a timeline format or a tabular (revision log) format.

code (library and applications) repository timeline http://trac.nexusformat.org/code/timeline

definitions repository timeline http://trac.nexusformat.org/definitions/timeline

code repository revision log http://trac.nexusformat.org/code/log

definitions repository revision log http://trac.nexusformat.org/definitions/log

Login

To update files in these repositories you will need to use a subversion client such as *TortoiseSVN* (http://tortoisesvn.tigris.org) for Microsoft Windows or `svn` for command-line shells and also provide your NeXus Wiki username and password. Note that for subversion write access:

- If your Wiki username contains a space, write it with a space (i.e. do not replace the space with an _ as is done in WIKI URLs)
- You cannot use a *temporary password* (i.e. one that was emailed to you in response to a request). You must first log into MediaWiki with the temporary password and then go to your account section (http://www.nexusformat.org/Special:Preferences) and change the password.
- Your Wiki account must have an email address associated with it and this address must have been validated. To provide and/or validate your email address, log in and go to your account section (http://www.nexusformat.org/Special:Preferences).
- If you have login problems and have not changed your WIKI password since 20th October 2006, please go to the NeXus wiki login page (http://www.nexusformat.org/Special:UserLogin) and request a new password to be sent by email. To synchronise TRAC/Subversion/MediaWiki required some changes to the authentication system which will have invalidated passwords set prior to that date.

Here are the URLs to access the subversion repositories as a developer:

code for library/applications https://svn.nexusformat.org/code/trunk

definitions for NXDL classes https://svn.nexusformat.org/definitions/trunk

checkout the code trunk

```
svn co --username "use your WIKI Username" https://svn.nexusformat.org/code/trunk nexus_code
```

Please report any problems via the *Issue Reporting* system.

Committing Changes

As well as needing a valid account, you will not be able to check-in changes unless you indicate (in the log message attached to the commit) which current issues on the *Issue Reporting* system the changes either fix or refer to. This is done by enclosing special phrases in the commit message of the form:

```
command #1
command #1, #2
command #1 & #2
command #1 and #2
```

where `command` is one of the commands detailed below and `#1` means *issue number 1* on the system, etc. You can have more then one command in a message. The following commands are supported and there is more then one spelling for each command (to make this as user-friendly as possible):

`closes, fixes` The specified issue numbers are closed with the contents of this commit message being added to it.

references, refs, addresses, re The specified issue numbers are left in their current status, but the contents of this commit message are added to their notes.

For example, the commit message

```
Changed blah and foo to do this or that. Fixes #10 and #12, and refs #12.
```

This will close issues #10 and #12, and add a note to #12 on the *Issue Reporting* system. For a list of current issues, see:

- **Active tickets for the NeXus code library:** http://trac.nexusformat.org/code/report/1
- **Active tickets for NeXus definitions:** http://trac.nexusformat.org/definitions/report/1

URLs described in this section

Many Uniform Resource Locators (URLs) have been used in this section. This is a table describing them.

Subversion revision management software http://subversion.apache.org/

ViewVC versions control repository viewing software http://www.viewvc.org/

TRAC issue management software http://trac.edgewall.org

TortoiseSVN, Windows subversion client http://tortoisesvn.tigris.org/

NeXus code (library and applications) subversion repository http://svn.nexusformat.org/code/

NeXus definitions subversion repository http://svn.nexusformat.org/definitions/

ViewVC view of NeXus code (library and applications) repository http://svn.nexusformat.org/viewvc/NeXusCode

ViewVC view of NeXus definitions repository http://svn.nexusformat.org/viewvc/NeXusDefinitions

TRAC view of NeXus code (library and applications) repository http://trac.nexusformat.org/code/browser

NeXus code (library and applications) revision log http://trac.nexusformat.org/code/log

Active tickets for the NeXus code repository http://trac.nexusformat.org/code/report/1

NeXus code repository timeline http://trac.nexusformat.org/code/timeline

TRAC view of NeXus definitions repository http://trac.nexusformat.org/definitions/browser

NeXus definitions revision log http://trac.nexusformat.org/definitions/log

Active tickets for NeXus definitions http://trac.nexusformat.org/definitions/report/1

NeXus definitions repository timeline http://trac.nexusformat.org/definitions/timeline

NeXus code repository (password required) https://svn.nexusformat.org/code/trunk

NeXus definitions repository (password required) https://svn.nexusformat.org/definitions/trunk

4.3.4 NeXus Issue Reporting

NeXus is using *TRAC* (http://trac.edgewall.org) for problem/issue reporting. The issue reports (see *View current issues* below) are used to guide the NeXus developers in resolving problems as well as implementing new features. As such, the TRAC tickets for the *code* and *definitions* repositories form the basis of a *roadmap* for NeXus. You can browse issues without logging on, but to report issues you will need to login using your NeXus WIKI username and password (the *subversion login notes* mentioned for write access to the *Subversion Server* apply to TRAC login, too).

Whenever an update is made to a ticket, a message is also posted to the appropriate *ticket mailing list*.

NeXus Code (NAPI, Library, and Applications)

Report a new issue http://trac.nexusformat.org/code

View current issues http://trac.nexusformat.org/code/report/1

Archive of ticket update emails http://lists.nexusformat.org/pipermail/nexus-code-tickets/

repository timeline (recent ticket and code changes) http://trac.nexusformat.org/code/timeline

repository roadmap http://trac.nexusformat.org/code/roadmap

NeXus Definitions (NXDL base classes and application definitions)

Report a new issue http://trac.nexusformat.org/definitions

View current issues http://trac.nexusformat.org/definitions/report/1

Archive of ticket update emails http://lists.nexusformat.org/pipermail/nexus-definitions-tickets/

repository timeline (recent ticket and definition changes) http://trac.nexusformat.org/definitions/timeline

repository roadmap http://trac.nexusformat.org/definitions/roadmap

CHAPTER

FIVE

RELEASES

The release history of the NeXus code repository (including the NAPI and applications) and the NeXus definitions repository (the NXDL and the manual) will be listed here. A full listing of the NeXus various repositories and other resources is given in the section titled *NeXus Subversion Repositories*.

5.1 NeXus code repository releases

Here will be listed the subversion repository URLs for releases of the NeXus code repository, including the NAPI and applications.

-tba-

5.2 NeXus definitions repository releases

Here are listed the subversion repository URLs for releases of the NeXus definitions repository, including the NXDL and the manual.

3.1.0 2012-05-14: http://svn.nexusformat.org/definitions/tags/v3.1.0/

CHAPTER

SIX

INSTALLATION

This section describes how to install the NeXus API and details the requirements. The NeXus API is distributed under the terms of the *LGPL: GNU Lesser Gnu Public License*.

The source code and binary versions for some popular platforms can be found on http://download.nexusformat.org/kits/. Up to date instructions can be found on the *NeXus Wiki* Download page (http://www.nexusformat.org/Download). In case you need help, feel free to contact the NeXus mailing list: http://lists.nexusformat.org/mailman/listinfo/nexus

6.1 Precompiled Binary Installation

6.1.1 Prerequisites

HDF5/HDF4

Note: HDF5 is the preferred format to use for NeXus.

NeXus uses HDF5 as the main underlying binary format. (HDF4 is supported as a legacy underlying binary format but is not recommended for new use.) It is necessary first to install the HDF subroutine libraries and include files before compiling the NeXus API. It is not usually necessary to download the HDF source code since precompiled object libraries exist for a variety of operating systems including Windows, Mac OS X, Linux, and various other flavors of Unix. Check the HDF web pages for more information: http://www.hdfgroup.org/

Packages for HDF4 and HDF5 are available for both Fedora (hdf, hdf5, hdf-devel, hdf5-devel) and Ubuntu/Debian (libhdf4g, libhdf5).

XML

Note: XML is not the preferred format to use for NeXus.

The NeXus API also supports using XML as a legacy underlying on-disk format. This uses the Mini-XML library, developed by Michael Sweet, which is also available as a precompiled binary library for several operating systems. Check the Mini-XML web pages for more information: http://www.minixml.org/

Packages for MXML are available for both Fedora (mxml, mxml-devel) and Ubuntu/Debian (libmxml1).

6.1.2 Linux RPM Distribution Kits

An installation kit (source or binary) can be downloaded from: http://download.nexusformat.org/kits/

A NeXus binary RPM (nexus-*.i386.rpm) contains ready compiled NeXus libraries whereas a source RPM (nexus-*.src.rpm) needs to be compiled into a binary RPM before it can be installed. In general, a binary RPM is installed using the command

```
rpm -Uvh file.i386.rpm
```

or, to change installation location from the default (e.g. /usr/local) area, using

```
rpm -Uvh --prefix /alternative/directory file.i386.rpm
```

If the binary RPMS are not the correct architecture for you (e.g. you need x86_64 rather than i386) or the binary RPM requires libraries (e.g. HDF4) that you do not have, you can instead rebuild a source RPM (.src.rpm) to generate the correct binary RPM for you machine. Download the source RPM file and then run

```
rpmbuild --rebuild file.src.rpm
```

This should generate a binary RPM file which you can install as above. Be careful if you think about specifying an alternative buildroot for rpmbuild by using `--buildroot` option as the "buildroot" directory tree will get remove (so `--buildroot /` is a really bad idea). Only change buildroot it if the default area turns out not to be big enough to compile the package.

If you are using Fedora, then you can install all the dependencies by typing

```
yum install hdf hdf-devel hdf5 hdf5-devel mxml mxml-devel
```

6.1.3 Microsoft Windows Installation Kit

A Windows MSI based installation kit is available and can be downloaded from: http://download.nexusformat.org/kits/windows/

6.1.4 Mac OS X Installation Kit

An installation disk image (.dmg) can be downloaded from: http://download.nexusformat.org/kits/macosx/

6.2 Source Installation

6.2.1 NeXus Source Code Distribution

The build uses `autoconf` (so autools are required) to determine what features will be available by your system. You must have the *development* libraries installed for all the file backends you want support for (see above). If you intend to build more than the C language bindings, you need to have the respective build support in a place where autoconf will pick them up (i.e. python development files, a Java Development Kit, etc.).

For more information see the README in the toplevel of the source distribution. In case you need help, feel free to contact the *NeXus Mailing List*:

Archives http://lists.nexusformat.org/mailman/listinfo/nexus

email nexus@nexusformat.org

Download the appropriate gzipped tar file, unpack it, and run the standard configure procedure from the resulting nexus directory. For example, for version 4.2.1;

```
$ tar zxvf nexus-4.2.1.tar.gz
$ cd nexus-4.2.1
$ ./configure
```

To find out how to customize the installation, e.g., to choose different installation directories, type

```
$ ./configure --help
```

Carefully check the final output of the `configure` run. Make sure all features requested are actually enabled.

```
$ make
$ make install
```

See the README file for further instructions.

6.2.2 Cygwin Kits

HDF4 is not supported under CYGWIN - both HDF5 and MXML are supported and can be downloaded and built as usual. When configuring HDF5 you should explicitly pass a prefix to the configure script to make sure the libraries are installed in a "usual" location i.e.

```
./configure --prefix=/usr/local/hdf5
```

Otherwise you will have to use the `--with-hdf5=/path/to/hdf5` option later when configuring NeXus to tell it where to look for hdf5. After building hdf5, configure and build NeXus using the instructions for source code distribution above.

CHAPTER

SEVEN

NEXUS UTILITIES

There are many utilities available to read, browse, write, and use NeXus data files. Some are provided by the NeXus technical group while others are provided by the community. Still, other tools listed here can read or write one of the low-level file formats used by NeXus (HDF4, HDF5, or XML).

7.1 Utilities supplied with NeXus

Most of these utility programs are run from the command line. It will be noted if a program provides a graphical user interface (GUI). Short descriptions are provided here with links to further information, as available.

nxbrowse NeXus Browser

nxconvert Utility to convert a NeXus file into HDF4/HDF5/XML/...

nxdir `nxdir` is a utility for querying a NeXus file about its contents. Full documentation can be found by running this command:

```
nxdir -h
```

nxingest

`nxingest` extracts the metadata from a NeXus file to create an XML file according to a mapping file. The mapping file defines the structure (names and hierarchy) and content (from either the NeXus file, the mapping file or the current time) of the output file. See the man page for a description of the mapping file. This tool uses the NAPI. Thus, any of the supported formats (HDF4, HDF5 and XML) can be read.

nxsummary Use `nxsummary` to generate summary of a NeXus file. This program relies heavily on a configuration file. Each `item` tag in the file describes a node to print from the NeXus file. The `path` attribute describes where in the NeXus file to get information from. The `label` attribute will be printed when showing the value of the specified field. The optional `operation` attribute provides for certain operations to be performed on the data before printing out the result. See the source code documentation for more details.

nxtranslate `nxtranslate` is an anything to NeXus converter. This is accomplished by using translation files and a plugin style of architecture where `nxtranslate` can read from new formats as plugins become available. The documentation for `nxtranslate` describes its usage by three types of individuals:

- the person using existing translation files to create NeXus files
- the person creating translation files
- the person writing new *retrievers*

All of these concepts are discussed in detail in the documentation provided with the source code.

nxvalidate From the source code documentation:

"Utility to convert a NeXus file into HDF4/HDF5/XML/..."

Note: this command-line tool is different than the newer Java GUI program: `NXvalidate`.

NXvalidate Java program (in development in 2010) to check any NeXus data file for conformance with the NeXus NXDL-based standard. Note: This Java GUI is different than the command-line tool: `nxvalidate`.

NXplot An extendable utility for plotting any NeXus file. `NXplot` is an Eclipse-based GUI project in Java to plot data in NeXus files. (The project was started at the first NeXus Code Camp in 2009.)

7.2 Data Analysis

The list of applications below are some of the utilities that have been developed (or modified) to read/write NeXus files as a data format. It is not intended to be a complete list of all available packages.

DAVE (http://www.ncnr.nist.gov/dave/) DAVE is an integrated environment for the reduction, visualization and analysis of inelastic neutron scattering data. It is built using IDL (Interactive Data Language) from ITT Visual Information Solutions.

GDA (http://www.opengda.org) The GDA project is an open-source framework for creating customised data acquisition and analysis software for science facilities such as neutron and X-ray sources.

Gumtree (http://docs.codehaus.org/display/GUMTREE) Gumtree is an open source project, providing a graphical user interface for instrument status and control, data acquisition and data reduction.

ISAW (ftp://ftp.sns.gov/ISAW/) The Integrated Spectral Analysis Workbench software project (ISAW) is a Platform-Independent system Data Reduction/Visualization. ISAW can be used to read, manipulate, view, and save neutron scattering data. It reads data from IPNS run files or NeXus files and can merge and sort data from separate measurements.

LAMP (http://www.ill.eu/data_treat/lamp/>) LAMP (Large Array Manipulation Program) is designed for the treatment of data obtained from neutron scattering experiments at the Institut Laue-Langevin. However, LAMP is now a more general purpose application which can be seen as a GUI-laboratory for data analysis based on the IDL language.

Mantid (http://www.mantidproject.org/) The Mantid project provides a platform that supports high-performance computing on neutron and muon data. It is being developed as a collaboration between Rutherford Appleton Laboratory and Oak Ridge National Laboratory.

NeXpy (http://nexpy.github.io/nexpy/) The goal of NeXpy is to provide a simple graphical environment, coupled with Python scripting capabilities, for the analysis of X-Ray and neutron scattering data. (It was decided at the NIAC 2010 meeting that a large portion of this code would be adopted in the future by NeXus and be part of the distribution)

OpenGENIE (http://www.opengenie.org/) A general purpose data analysis and visualisation package primarily developed at the ISIS Facility, Rutherford Appleton Laboratory.

PyMCA (http://pymca.sourceforge.net/) PyMca is a ready-to-use, and in many aspects state-of-the-art, set of applications implementing most of the needs of X-ray fluorescence data analysis. It also provides a Python toolkit for visualization and analysis of energy-dispersive X-ray fluorescence data. Reads, browses, and plots data from NeXus HDF5 files.

7.3 HDF Tools

Here are some of the generic tools that are available to work with HDF files. In addition to the software listed here there are also APIs for many programming languages that will allow low level programmatic access to the data structures.

HDF Group command line tools (http://www.hdfgroup.org/products/hdf5_tools/#h5dist/) There are various command line tools that are available from the HDF Group, these are usually shipped with the HDF5 kits but are also available for download separately.

HDFexplorer (http://www.space-research.org/) A data visualization program that reads Hierarchical Data Format files (HDF, HDF-EOS and HDF5) and also netCDF data files.

HDFview (http://www.hdfgroup.org) A Java based GUI for browsing (and some basic plotting) of HDF files.

IDL (http://www.ittvis.com/) IDL is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation.

IgorPro (http://www.wavemetrics.com/) IGOR Pro is an extraordinarily powerful and extensible scientific graphing, data analysis, image processing and programming software tool for scientists and engineers.

MATLAB (http://www.mathworks.com/) MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation.

7.3.1 Language APIs

h5py (http://code.google.com/p/h5py/) HDF5 for Python (h5py) is a general-purpose Python interface to HDF5.

CHAPTER

EIGHT

FREQUENTLY ASKED QUESTIONS

This is a list of commonly asked questions concerning the NeXus data format.

1. Is it Nexus, NeXus or NeXuS?

   NeXus is correct. It is a format for data from **Neutron** and **X-ray** facilities, hence those first letters are capitalised. The format is also used for muon experiments, but there is no *mu* (or m) in NeXus and no s in muon. So the s stays in lower case.

2. How many facilities use NeXus?

   This is not easy to say, not all facilities using NeXus actively participate in the committee. Some facilities have reported their adoption status on the Facilities Wiki page. Please have a look at this list. Keep in mind that it is not complete.

3. NeXus files are binary? This is crazy! How am I supposed to see my data?

   NeXus files are not per se binary. If you use the XML backend the data are stored in a relatively human readable form (see this example). This backend however is only recommended for very small data sets. With the multidimensional data that is routinely recorded on many modern instruments it is very difficult anyway to retrieve useful information on a VT100 terminal. If you want to try, for example `nxbrowse` is a utility provided by the NeXus community that can be very helpful to those who want to inspect their files and avoid graphical applications. For larger data volumes the binary backends used with the appropriate tools are by far superior in terms of efficiency and speed and most users happily accept that after having worked with supersized "human readable" files for a while.

4. What on-disk file format should I choose for my data?

   HDF5 is the default file container to use for NeXus data. It is the recommended format for all applications. HDF4 is still supported as a on disk format for NeXus but for new installations preference should be given to HDF5. The XML backend is available for special use cases. Choose this option with care considering the space and speed implications.

5. Why are the NeXus classes so complicated? I'll never store all that information

   The NeXus classes are essentially glossaries of terms. If you need to store a piece of information, consult the class definitions to see if it has been defined. If so, use it. It is not compulsory to include every item that has been defined in the base class if it is not relevant to your experiment. On the other hand, a NeXus application definition lists a smaller set of compulsory items that should allow other researchers or software to analyze your data. You should really follow the application definition that corresponds to your experiment to take full advantage of NeXus.

6. I don't like NeXus. It seems much faster and simpler to develop my own file format. Why should I even consider NeXus?

   If you consider using an efficient on disk storage format, HDF5 is a better choice than most others. It is fast and efficient and well supported in all mainstream programming languages and a fair share

of popular analysis packages. The format is so widely used and backed by a big organisation that it will continue to be supported for the foreseeable future. So if you are going to use HDF5 anyway, why not use the NeXus definition to lay out the data in a standardised way? The NeXus community spent years trying to get the standard right and while you will not agree with every single choice they made in the past, you should be able to store the data you have in a quite reasonable way. If you do not comply with NeXus, chances are most people will perceive your format as different but not necessarily better than NeXus by any large measure. So it may not be worth the effort. Seriously.

If you encounter any problems because the classes are not sufficient to describe your configuration, please contact the NIAC Executive Secretary explaining the problem, and post a suggestion at the relevant class wiki page. Or raise the problem in one of the mailing lists. The NIAC is always willing to consider new proposals.

7. I want to produce an application definition. How do I go about it?

   Read the NXDL Tutorial in *Creating a NXDL Specification* and have a try. You can ask for help on the mailing lists. Once you have a definition that is working well for at least your case, you can submit it to the NIAC for acceptance as a standard. The procedures for acceptance are defined in the NIAC constitution. [1]

8. What is the purpose of `NXdata`?

   `NXdata` contains links to the data stored elsewhere in the `NXentry`. It identifies the default plottable data. This is one of the basic motivations (see *Simple plotting*) for the NeXus standard. The choice of the name `NXdata` is historic and does not really reflect its function.

9. How do I identify the plottable data?

   See the section: *Find the plottable data*.

10. How can I specify reasonable axes for my data?

    See the section: *Linking Multi Dimensional Data with Axis Data*.

11. Why aren't `NXsample` and `NXmonitor` groups stored in the `NXinstrument` group?

    A NeXus file can contain a number of `NXentry` groups, which may represent different scans in an experiment, or sample and calibration runs, etc. In many cases, though by no means all, the instrument has the same configuration so that it would be possible to save space by storing the `NXinstrument` group once and using multiple links in the remaining `NXentry` groups. It is assumed that the sample and monitor information would be more likely to change from run to run, and so should be stored at the top level.

12. Specifications are complicated and often provide too much information for what I need. Where can I find some good example data files?

    There are a few checked into the definitions repository. At the moment the selection is quite limited and not very representative. This repository will be edited as more example files become available.

13. Can I use a NXDL specification to parse a NeXus data file?

    This should be possible as there is nothing in the NeXus specifications to prevent this but it is not implemented in NAPI. You would need to implement it for yourself. You would be wise to consult the algorithms in the Java version of `NXvalidate` (see the *Java-version of NXvalidate*) for more details.

14. Why do I need to specify the `NAPItype`? My programming language does not need that information and I don't care about C and colleagues. Can I leave it out?

[1] Refer to the most recent version of the NIAC constitution on the NIAC wiki: http://www.nexusformat.org/NIAC

`NAPItype` is necessary. When implementing the NeXus-XML API we strived to make this as general as HDF and reasonably efficient for medium sized datasets. This is why we store arrays as a large bunch of numbers in C-storage order. And we need the `NAPItype` to figure out the dimensions of the dataset.

15. Do I have to use the `NAPI` subroutines? Can't I read (or write) the NeXus data files with my own routines?

    You are not required to use the NAPI to write valid NeXus data files. It is possible to avoid the NAPI to write and read valid NeXus data files. But, the programmer who chooses this path must have more understanding of how the NeXus HDF or XML data file is written. Validation of data files written without the NAPI is strongly encouraged.

16. I'm using links to place data in two places. Which one should be the data and which one is the link?

    Note: NeXus uses HDF5 hard links

    In HDF, a hard link points to a data object. A soft link points to a directory entry. Since NeXus uses hard links, there is no need to distinguish between two (or more) directory entries that point to the same data.

    Both places have pointers to the actual data. That is the way hard links work in HDF5. There is no need for a preference to either location. NeXus defines a `target` attribute to label one directory entry as the source of the data (in this, the link *target*). This has value in only a few situations such as when converting the data from one format to another. By identifying the original in place, duplicate copies of the data are not converted.

17. **If I write my data according to the current specification for *NXsas*** (substitute any other application definition), will other software be able to read my data?

    Yes. *NXsas*, like other *Application Definitions*, defines and names the *minimum information* required for analysis or data processing. As long as all the information required by the specification is present, analysis software should be able to process the data. If other information is also present, there is no guarantee that small-angle scattering analysis software will notice.

18. Where do I store the wavelength of my experiment?

    See the *Strategies: The wavelength* section.

CHAPTER

NINE

BRIEF HISTORY OF THE NEXUS FORMAT

Two things to note about the development and history of NeXus:

- All efforts on NeXus have been voluntary except for one year when we had one full-time worker.
- The NIAC has already discussed many matters related to the format.

June 1994 Mark Könnecke (then ISIS, now PSI) made a proposal using netCDF [1] for the European neutron scattering community while working at ISIS

August 1994 Jonathan Tischler (ORNL) proposed an HDF-based format [2] as a standard for data storage at APS

October 1994 Ray Osborn convened a series of three workshops called *SoftNeSS*. [3] In the first meeting, Mark Könnecke and Jon Tischler were invited to meet with representatives from all the major U.S. neutron scattering laboratories at Argonne National Laboratory to discuss future software development for the analysis and visualization of neutron data. One of the main recommendations of *SoftNeSS'94* was that a common data format should be developed.

September 1995 At *SoftNeSS 1995* (at NIST), three individual data format proposals by Przemek Klosowski (NIST), Mark Könnecke (then ISIS), and Jonathan Tischler (ORNL and APS/ANL) were joined to form the basis of the current NeXus format. At this workshop, the name *NeXus* was chosen.

August 1996 The HDF-4 API is quite complex. Thus a NeXus Abstract Programmer Interface (NAPI) EDIT_ME was released which simplified reading and writing NeXus files.

October 1996 At *SoftNeSS 1996* (at ANL), after reviewing the different scientific data formats discussed, it was decided to use HDF-4 as it provided the best grouping support. The basic structure of a NeXus file was agreed upon. the various data format proposals were combined into a single document by Przemek Klosowski (NIST), Mark Könnecke (then ISIS), Jonathan Tischler (ORNL and APS/ANL), and Ray Osborn (IPNS/ANL) coauthored the first proposal for the NeXus scientific data standard. [4]

July 1997 SINQ at PSI started writing NeXus files to store raw data.

Summer 2001 MLNSC at LANL started writing NeXus files to store raw data

September 2002 NeXus API version 2.0.0 is released. This version brought support for the new version of HDF, HDF-5, released by the HDF group. HDF-4 imposed limits on file sizes and the number of objects in a file. These issues were resolved with HDF-5. The NeXus API abstracted the difference between the two physical file formats away form the user.

[1] http://wiki.nexusformat.org/images/b/b8/European-Formats.pdf
[2] http://www.neutron.anl.gov/softness
[3] http://wiki.nexusformat.org/images/d/d5/Proposed_Data_Standard_for_the_APS.pdf
[4] http://wiki.nexusformat.org/images/9/9a/NeXus_Proposal.pdf

June 2003 Przemek Klosowski, Ray Osborn, and Richard Riedel received the only known grant explicitly for working on NeXus from the Systems Integration for Manufacturing Applications (SIMA) program of the National Institute of Standards and Technology (NIST). The grant funded a person for one year to work on community wide infrastructure in NeXus.

October 2003 In 2003, NeXus had arrived at a stage where informal gatherings of a group of people were no longer good enough to oversee the development of NeXus. This lead to the formation of the NeXus International Advisory Committee (NIAC) which strives to include representatives of all major stake holders in NeXus. A first meeting was held at CalTech. Since 2003, the NIAC meets every year to discuss all matters NeXus.

July 2005 The community asked the NeXus team to provide an ASCII based physical file format which allows them to edit their scientific results in emacs. This lead to the development of a XML NeXus physical format. This was released with NeXus API version 3.0.0.

May 2007 NeXus API version 4.0.0 is released with broader support for scripting languages and the feature to link with external files.

October 2007 NeXus API version 4.1.0 is released with many bug-fixes.

October 2008 *NXDL: The NeXus Definition Language* is defined. Until now, NeXus used another XML format, meta-DTD, for defining base classes and application definitions. There were several problems with meta-DTD, the biggest one being that it was not easy to validate against it. NXDL was designed to circumvent these problems. All current base classes and application definitions were ported into the NXDL.

April 2009 NeXus API version 4.2.0 is released with additional C++, IDL, and python/numpy interfaces.

September 2009 NXDL and draft `NXsas` presented to canSAS at SAS2009 conference

January 2010 NXDL presented to ESRF HDF5 workshop on hyperspectral data

May 2012 first release (3.1.0) of NXDL (NeXus Definition Language)

9.1 Historical notes about the Development of NXDL

This section contains a few brief notes about the history of NXDL and the motivations for its creation.

Previously, the structure of NeXus data files was described using *Meta-DTD*, an XML format that provided a compact description. The terse format was not obvious to all and was difficult to machine-process. NXDL was conceived to be a simpler syntax than Meta-DTD. The switch to NXDL was not intended to change what was in the data files, just to provide an easier (and more generic) way of describing data files.

The NeXus Design page lists the group classes from which a NeXus file is constructed. They provide the glossary of items that could, in principle, be stored in a standard-conforming NeXus file (other items may be inserted into the file if the author wishes, but they won't be part of the standard). When planning to include a particular piece of metadata, consult the class definitions to find out what to call it. However, to assist those writing data analysis software, it is useful to provide more than a glossary; it is important to define the required contents of NeXus files that contain data from particular classes of neutron, x-ray, or muon instrument.

As part of the NeXus standard, the NIAC identified a number of generic instruments that describe an appreciable number of existing instruments around the world. Although not identical in every detail, they share many common characteristics, and more importantly, they require sufficiently similar modes of data analysis, enough to make a standard description useful. Many of the application definitions were built from these instrument definitions using the NeXus Definition Language (NXDL) format.

Class definitions in NeXus prior to 2008 had been in the form of base classes and instrument definitions. All of these were in the same category. As the development of NeXus had been led mostly by scientists from neutron sources, this represented their typical situations.

Both those new to NeXus and also those familiar saw the previous emphasis on instrument definitions as a deficiency that limited flexibility and possibly usage. The point was made that NeXus should attempt to describe better reduced data and also data for analysis since synchrotron instruments are rarely adhering to a fixed definition.

The design of NeXus is moving towards an object-oriented approach where the base classes will be the objects and the application definitions will use the objects to specify the required components as fits some application. Here, *application* is very loosely defined to include:

- specification of a scientific instrument (example: TOF-USANS at SNS)
- specification of what is expected for a scientific technique (example: small-angle scattering data for common analysis programs)
- specification of generic data acquisition stream (example: TOFRAW - raw time-of-flight data from a pulsed neutron source)
- specification of input or output of a specific software program

The point of the *NeXus Application Definition* is that all of these start with `NX` and all have been approved by the NIAC.

Those NXDL specifications not yet approved by the NIAC fall into the category of *NeXus contributed definitions* for which NeXus has a place in the repository. Consider the NXDL files in the `contributed` directory as *in incubation*. This category is the place to put an NXDL (a candidate for a base class or application definition) for the NIAC to consider approving.

CHAPTER

TEN

REVISION HISTORY

date	release	description	initials
2012-09		Documentation converted from DocBook to Sphinx. Manual back to a single volume.	PRJ
2012-05	3.1	Ready for release.	PRJ
2012-02		Now using cmake to control multiplatform build and packaging.	PRJ
2011-11	1.0b	Preparing manual for initial release. Also preparing to convert manual source from DocBook to Sphinx for next release of manual.	PRJ
2010-11	draft	Nearly complete but still much finishing work remains. The description of dimensions and the description of the coordinate system needs major revision and improvement. More examples are needed. The manual is now divided into two volumes. Volume I is the User Manual, Volume II is the Reference Documentation. Much of the NXDL chapter in Volume II is autogenerated from the `nxdl.xsd` Schema and the NXDL source files. Initial release of NXDL, manual, and next release of NAPI (compatibility release) expected in mid-2011.	PRJ
2010 spring	initial draft	Most of the content from the old NeXus mediawiki documentation is included. Some new wiki content has been introduced but should be easy to identify for inclusion in the manual.	PRJ
2009-11		Started conversion from the old NeXus mediawiki documentation.	PFP

CHAPTER

ELEVEN

COPYRIGHT

2013-11-25 20:20:44 GMT

Published 2011 by NeXus International Advisory Committee, http://www.nexusformat.org

11.1 Licenses

11.1.1 FDL: GNU Free Documentation License

The NeXus manual is licensed under the terms of the GNU Free Documentation License version 1.3.

download `FDL`

GNU http://www.gnu.org/licenses/fdl-1.3.txt

11.1.2 LGPL: GNU Lesser Gnu Public License

The examples in the NeXus manual are licensed under the terms of the GNU Lesser General Public License version 3.

download `LGPL`

GNU http://www.gnu.org/licenses/lgpl-3.0.txt

CHAPTER

TWELVE

ABOUT THIS MANUAL

This manual was produced using Sphinx (http://sphinx.pocoo.org). The source for the manual shows many examples of the structures used to create the manual. If you have any questions about how to contribute to this manual, please contact the NeXus Documentation Editor (Pete Jemian <jemian@anl.gov>).

Note: The indentation is very important to the syntax of the restructured text manual source. Be careful not to mix tabs and spaces in the indentation or the manual may not build properly.

Publishing Information

This manual built 2013-11-25 20:20:47 GMT.

See Also:

This document is available in different formats:

online HTML http://download.nexusformat.org/doc/html/index.html

PDF `nexus.pdf` (available via online HTML link above)

A very brief overview is also available (separate from the manual).

HTML http://svn.nexusformat.org/definitions/trunk/misc/impatient/_build/html/index.html

INDEX

N