
# NeXus for the Impatient

***Release 2022***

**nexusformat.org**

**May 21, 2025**

## Contents


# 1 Why NeXus?

**See also**

This document is available online in HTML and PDF formats.

The NeXus data format[1] is a tool which has been designed to solve the problems of travelling scientists, who undertake experiments at several different neutron, x-ray or muon facilities. Such people will apply to different facilities with different proposals in order to get their science done, then combine the data thus collected at the various sources. However, the life of a travelling scientist is complicated by some of the following factors:

- They will find several different file formats at each facility, some of which may even be proprietary formats which only can be read by expensive tools.
- They waste time writing programs to get the data into their favourite data analysis software.
- They may need many different files in different formats, plus local knowledge and notes in order to analyse data.
- They have to deal with inefficient data formats.
- They cannot easily read their collaborators' data.

NeXus is designed to solve these problems by defining a data format with the following properties:

**self-describing**
: The content of the file can be discovered without prior knowledge.

**extendable**
: This means that you can always add more data to the file.

**platform independent**
: It must be readable on UNIX, Macintosh, Windows or whatever computers.

**public domain**
: Both the file format specifications and the API to access the file data must be in the public domain.

**efficient**
: Data from modern high speed detectors should be both written and read quickly.

[1] NeXus: https://www.nexusformat.org/

**complete**
All data required for typical usage cases should be contained in one file; even better, a full beam line description.

**flexible**
In order to be applicable to a wild variety of applications.

# 2 NeXus Concepts

NeXus uses HDF-5[2] files as container files. HDF-5 is a popular scientific data format developed by the National Center for Supercomputing Applications (NCSA), University of Illinois Urbana-Champaign and currently maintained by *The HDF Group*. There is built-in support for HDF-5 in many scientific packages. Other users of HDF-5 include NASA, Boeing, meteorological offices around the world and many more. NeXus is thus able to inherit many desirable properties for free from HDF-5, such as: extendable, self-describing, platform independent, public domain, and efficient. For historical reasons, NeXus supports two further container file formats: HDF-4 and XML. The NeXus use of these formats is now deprecated.

To understand NeXus, it is important to know about some of the objects encountered in an HDF-5 file:

**groups**
Allow structuring of information in a file. Groups work pretty much like folders/directories in a file system.

**datasets**
These are multidimensional arrays of numbers and character strings.

**attributes**
These are little morsels of metadata which can be attached to either groups or datasets.

**links**
These are pointers to HDF-5 objects which live in a different place in the hierarchy. They remove the need to copy data when a given item needs to be referenced at multiple places in an HDF-5 hierarchy. HDF-5 Links behave like symbolic links in a UNIX file system.

HDF-5 does not, however, know anything about the application domain of neutron, muon or X-ray scattering. In order to remedy this, NeXus adds the following:

**A group hierarchy in the HDF-5 file**
A group hierarchy in the files helps with a couple of issues. If a full beamline description is stored, then this can easily be hundreds of parameters - a hierarchy brings some order into a potential mess. The hierarchy also allows us to store more then one experiment and/or result of data analysis in the same file. Thus a NeXus file may contain a complete scientific workflow from raw data to publishable results.

**Rules for storing data in a file**
NeXus wants you to store physical values, with units and uncertainty, but the motor encoder positions, control voltages, set points, *etc.* can be stored as well. Each dataset should indicate the axes required to plot the data. By prescribing the structure of the data fields, we minimize the amount of variations and surprises when accessing NeXus files.

**A dictionary of field names**
Even with units and axes available, an item with a name like *stx* has little meaning. NeXus provides a dictionary of names applicable to neutron, x-ray or muon scattering and documents their meanings.

**Application Definitions**
In order to import data seamlessly into an application program the file producer and the file consumer have to agree what has to be in the NeXus file for a certain use case. This is the purpose of an application definition. Application definitions are used to define strict standards. NeXus files can be verified against application definitions with a special tool called nxvalidate.

[2] HDF-5: http://www.hdfgroup.org/HDF5/

**Tools**

In order to make dealing with HDF-5 files easier, NeXus has provided an API for reading and writing files for many programming languages. Moreover, NeXus has coded some tools for converting to and from NeXus as well as plotting and viewing utilities.

## 2.1 The NeXus File Hierarchy

NeXus defines two main group hierarchy types:

1. *The NeXus Raw Data File Hierarchy*
2. *The NeXus Processed Data Hierarchy*

There are additional hierarchy variations for multi-method instruments and for a general purpose dump structure. Documentation for these hierarchy types can be found in the NeXus manual.

### The NeXus Raw Data File Hierarchy

This hierarchy is applicable to raw data files as written by some facility instrument:

```
entry:NXentry
   @default = data
   data:NXdata
      @signal = data
      data --> /entry/instrument/detector/data
   instrument:NXinstrument
      source:NXsource
      ....
      detector:NXdetector
         data:NX_INT32[512,512]
   sample:NXsample
   control:NXmonitor
```

The following two groups are required for all NeXus data files:

```
entry:NXentry
   data:NXdata
```

**About the terms: `entry:NXentry`**

A NeXus group is shown with a two-part name, as in `entry:NXentry`. The first part, shown here as `entry` is the HDF5 group name and can be just about any text that is a valid variable name in most programming languages. See the NeXus manual[6] for full details and note that **no spaces are allowed**. The second part, shown here as `NXentry`, is the name of the NeXus class that describes the structure within this group.

**`entry:NXentry`**

At the top/root level of a NeXus file are the NXentry groups. Each entry represents a separate collection of datasets.

**`data:NXdata`**

This group is supposed to hold the most important data items of the experiment. This is a convenience so that a general plotting program can identify *from this group alone* what is the default data to render on a plot. Therefore the data is linked here from the specific detector entry recording the data which is described below. Axis information is usually included here as well, the details of which are covered in the NeXus manual.

[6] NeXus User Manual: https://manual.nexusformat.org/user_manual.html

The following additional groups are present in many NeXus data files:

`sample:NXsample`
: This group contains datasets which describe everything we know about the sample, including sample environment information such as temperature.

`instrument:NXinstrument`
: This group contains further groups and fields which describe the components of the instrument (i.e. beamline for synchroton sources) used for this experiment.

`control:NXmonitor`
: This group contains the counting information: which preset was used, how long we counted, monitor counts, etc.

**Note**

A few words on notation in this representation:

**indentation**
: Describes hierarchy level

`name:NXname`
: This describes a NeXus group. The second name, starting with `NX`, is the NeXus class name of the group. Each NeXus class defines a set of allowed field names that may be used to describe a component of the experiment, such as detector distance and angle. Not all such names are required - those relevant to a particular use case are specified by the appropriate application definition. Some experiments have multiple groups of the same class, such as apertures and detectors.

`name:NX_TYPE[dim,dim,...]`
: This describes a dataset with a given numeric type and dimensions. In this example, the detector data is a 512 x 512 array of 32-bit integers.

`@name=value`
: This describes an attribute name and value. Both groups and fields can have attributes. The attribute `signal=data` indicates to NeXus that the field in this group named `data` is the dependent data to be plotted.

`name --> path`
: Describes a link from one location to another. This allows us to gather the most important data together in an `NXdata` group while leaving detailed metadata in the individual component definitions. NeXus uses HDF5 hard links for pointing to other objects in the same file and external links when pointing to objects in other HDF5 files.

## 2.2 What goes into a NeXus File?

Before starting to describe how to decide what goes into a NeXus file, some more details about NeXus groups and base classes need to be explained. As seen in the examples, NeXus uses groups with well-defined class names starting with "NX". NeXus calls these NX classes "base classes", which is slightly misleading when you are used to object-oriented notations. For each NeXus base class, there exists a dictionary description that details which other groups and which fields are allowed in this base class. This dictionary is where you will find appropriate field names for the data items you wish to describe. The NeXus base classes are documented in the NeXus Reference Manual.[7] A common misconception among NeXus beginners is that you have to specify all fields which exist in a given NeXus base class. This is **not** the case! You only need to choose those fields from the NeXus base class dictionary which make sense for your application. But, you are encouraged to store additional, available information since it can be used to diagnose problems with the instrument. The minimum set of fields that are appropriate to a given technique are usually specifed in an "application definition".

[7] NeXus Reference Documentation: https://manual.nexusformat.org/ref_doc.html

Before the mechanics of writing a NeXus file can be explained, we need to know which fields are written into the NeXus file at which position in the hierarchy. The example will be to store basic data. Some steps are required:

Example 3: NeXus Raw Data File Template

```
entry:NXentry
   instrument:NXinstrument
   sample:NXsample
   control:NXmonitor
   data:NXdata
```

1. The start is a NeXus raw data file template as shown in example 3.
2. At this level you can decide what needs to be known about the sample and put it into the NXsample group.
3. Look at a design drawing of the instrument. For each major instrument component find a suitable NeXus class and add it to the NXinstrument group.
4. Decide for each instrument component which data fields are required and add them to the corresponding group.
5. Add required counting information to the control class.
6. Decide which data sets make up the most important data items in the experiment. Create links to these data items in the data group.
7. Investigate if a NeXus application definition exists for your instrument type. If so, check if all required fields are stored in the appropriate form.

Before beginning this process, it might be worthwhile to look at some of the NeXus application definitions in the NeXus reference manual for examples and inspiration. But be aware that each NeXus application definition only defines the minimum sets for a certain usage case.

In this process you might encounter the situation that you wish to store more information then foreseen by NeXus. There are two options which have to be considered:

1. The data item to store is special to your instrument and of no general interest. Then make up a name and store it. The beauty of NeXus is that this is possible without breaking the standard compliance of the file. Usual practice is to use a pattern like `facilityname_fieldname` which is unlikely to collide with fields that are added to the NeXus definition in the future.
2. The data item is of general interest and should be added to NeXus. Then suggest a name and document what this really is what you suggest. Forward this information to the NeXus International Advisory Committee (nexus-committee@nexusformat.org). Usually such suggestions are accepted quickly when they pose no conflicts with existing definitions.

Be sure that the names of things you define have no embedded whitespace and begin with a letter.

### The NeXus Processed Data Hierarchy

This is a simplified hierarchy style applicable to the results of data reduction or data analysis applications. Such results can consist of large multidimensional arrays, so it can be advisable to use NeXus for storing such data:

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

(continues on next page)

(continued from previous page)

```
   data:NXdata
      @signal = data
      data
```

Here the NXentry contains:

**@entry = data**
: Attribute on the NXentry group pointing to the default NXdata group to be plotted.

**data:NXdata**
: Contains the result of the data reduction directly, together with the axes required to use the data. The attribute @signal = data indicates that the field named data is the default plottable data.

**sample:NXsample**
: Contains the sample information. This may be a link to the sample information within a measurement entry elsewhere in the file.

**reduction:NXprocess**
: This group is used to document what kind of processing occurred to obtain the results stored in this NXentry. Here NeXus documents the name and version of the program used to do the reduction.

**input:NXparameters**
: The NXparameters groups describe the input and output parameters of the data reduction program. NeXus does not provide standard names here but rather provides containers to store this information which is important to make results reproducible.

Optionally, a processed data entry can contain an NXinstrument group in order to describe the instrument if this matters at this stage.

## 2.3 Scans in NeXus

Scanning means to vary some variable in a certain, defined way and collect data as the variable progresses. Scans are a versatile experimental technique and are thus very difficult to standardize. NeXus solves this problem through a couple of rules. Before these rules can be discussed, the symbol **NP** has to be introduced. NP is simply the number of scan points.

1. During a scan store each varied variable as an array of length NP at its appropriate place in the NeXus hierarchy.
2. For area detectors, the first (slowest varying) dimension becomes NP. Example: data from an area detector is stored as data[NP,xdim,ydim]
3. In NXdata, create links to all varied parameters and the detector data. Thus a representation similar to the conventional table representation of a scan is achieved.

This is an example of a NeXus raw data file describing a scan where the sample is rotated and data is collected in an area detector:

```
entry:NXentry
   @entry = data
   instrument:NXinstrument
      detector:NXdetector
         data:[NP,xsize,ysize]
   sample:NXsample
      rotation_angle[NP]
   control:NXmonitor
      data[NP]
   data:NXdata
```

(continues on next page)

(continued from previous page)

```
    @signal = 1
    @axes = ["rotation_angle", ".", "."]
    @rotation_angle_indices = 0
    data --> /entry/instrument/detector/data
    rotation_angle --> /entry/sample/rotation_angle
```

The default data to be plotted has more dimensions and requires additional description. The attribute `@axes = ["rotation_angle", ".", "."]` implies that the "signal data" (`data`) has three dimensions. The first dimension is provided by the `rotation_angle` field, while the `.` for the other two dimensions indicates they each should be plotted as index number (starting from zero). The attribute `@rotation_angle_indices = 0` declares that `rotation_angle` is used as the first dimension of `data`. The `AXISNAME_indices` attribute becomes more useful when both the "signal data" and the dimension scales are multi-dimensional.

## 2.4 Finding the plottable data

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

1. Start at the top level of the NeXus data file (the *root* of the HDF5 hierarchy).
2. Pick the default `NXentry` group, as designated by the `default` attribute.
3. Pick the default `NXdata` group, as designated by the `default` attribute.
4. Pick the default plottable field, as designated by the `signal` attribute.
    1. Pick the fields with the dimension scales (the `axes` attribute).
    2. Associate dimension scales with plottable data dimensions (the `AXISNAME_indices` attributes).
    3. Associate the dimension scales with each dimension of the plottable data.
5. Plot the *signal* data, given *axes* and *AXISNAME_indices*.

For details of this process, consult this section of the NeXus manual.[8]

# 3 NeXus Benefits

When trying to establish a data standard, we encounter a few challenges, some of which can slow effort:

**Science does new things**
: By definition, science is about doing new things, and of course new things cannot always be forced into strict standards. Thus any standardization effort in science will be an ongoing process.

**Consensus**
: In order to establish a standard, a large portion of a scientific community must agree.

**Resources**
: A data standard requires scientific programming resources to implement in acquisition and analysis software, but such resources are scarce especially at already established facilities.

However, there are many benefits to be gained from having the NeXus data standard:

**Discoverable format**
: By using a standard container format, people can examine their data from many software packages without any coding at all.

**NeXus dictionary**
: Using field names from a well documented dictionary gives meaning to the data in the file.

[8] Finding the plottable data: https://manual.nexusformat.org/datarules.html#version-3

**Programming**
Using suitable programming techniques a data processing program can read any NeXus file which contains the required data easily.

**Storing complete data**
Storing all this metadata when saving the data takes extra effort, but benefits include:

- The file will include the necessary fields for yet unforeseen ways to analyse the data.
- If something is wrong with the data, it becomes possible to figure out what went wrong.
- There is a better record of what has been measured. This helps to protect against scientific fraud.

**Application definitions**
For common measurement techniques with well-defined data reduction and analysis steps, data files with all the required fields included can be processed automatically. The NeXus application definitions serve the role of defining which fields are needed for a given measurement type.

# 4 Reading NeXus Files

The simplest way to read and plot a NeXus file is through the nexusformat[3] Python package :

```
from nexusformat.nexus import nxopen
nxopen('powder.h5').plot()
```

In order for this to be possible, *nexusformat* uses the NeXus conventions to locate the plottable data and the axes to use. In particular, this plots the first `NXdata` group in the first `NXentry` in the `powder.h5` file. The NeXus python package provides additional support for working with NeXus groups.

The plot could also be created by directly accessing the HDF-5 file using the h5py[4] package:

```
import pylab, h5py
file = h5py.File('powder.h5')
pylab.plot(file['/entry1/data1/two_theta'], file['/entry1/data1/counts'])
pylab.title(file['/entry1/title'][0])
pylab.show()
```

Matlab support in version R2011b is similar:

```
>> two_theta = h5read('powder.h5', '/entry1/data1/two_theta');
>> counts = h5read('powder.h5', '/entry1/data1/counts');
>> title = h5read('powder.h5', 'entry1/title');
>> plot(two_theta, counts)
>> title(title)
```

Note that matlab will require explicit casting from integer data to floating point data to perform many operations. For example, to plot a 2D data set[5] using log intensity:

```
>> data = h5read('lrcs3701.nx5','/Histogram1/data/data');
>> h = pcolor(log(double(data+1))); set(h,'EdgeAlpha',0)
```

Support for HDF is available in other scientific computing environments, including IDL, Igor, Mathematica and R.

Reading the file using the HDF-5 C API is a little more involved:

[3] *nexusformat*: https://nexpy.github.io/nexpy/
[4] *h5py*: https://www.h5py.org/
[5] `lrcs3701.nx5` (NeXus HDF-5 data file): https://github.com/nexusformat/exampledata/blob/master/IPNS/LRMECS/hdf5/lrcs3701.nx5

```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);
```

(continues on next page)

(continued from previous page)

```
  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]);
  }

}
```

**Note**

To keep these examples short, we already learned the HDF addresses of the "signal data" (such as `/entry1/data1/counts`) and its associated dimension scales (such as `/entry1/data1/two_theta`) for each example data file. The examples above use those addresses directly, rather than using the described method to find the plottable data.

If you are writing specific code to process a set of NeXus files where you already know the HDF5 addresses of the items you need, feel welcome to use that knowledge, as you have seen above.

If you are writing code to handle any NeXus data file, then you are advised to implement the methods to find the plottable data, as described in the manual.[Page 8, 8]

More examples of reading NeXus data files can be found in the *Examples* chapter of the NeXus Reference Documentation.[Page 5, 7]

# 5 Writing NeXus Files

You can obviously skip this section if you only wish to read NeXus files.

For writing the NeXus file, you have the option to use the NeXus API or to use the HDF-5 API. The complexity of NeXus file writing code is similar to the reading code. For both approaches, more information is available in the NeXus Manual[Page 4, 6] or the NeXus Reference Documentation.[Page 5, 7]

To give you a taste of what it is like to write a NeXus file using the NeXus API, here is a complete code example in C. It shows how to create a `scan:NXentry/data:NXdata` structure and store two arrays, counts and two_theta:

```
#include "napi.h"

int writer(float *tth, float *counts, int n)
{
   /* we receive two arrays: tth and counts, each length n*/
   NXhandle fileID;
   NXopen ("NXfile.nxs", NXACC_CREATE, &fileID);
      NXmakegroup (fileID, "Scan", "NXentry");
      NXopengroup (fileID, "Scan", "NXentry");
         NXmakegroup (fileID, "data", "NXdata");
         NXopengroup (fileID, "data", "NXdata");
```

(continues on next page)

(continued from previous page)

```
            NXputattr (fileID, "signal", "counts", 6, NX_CHAR);
            NXputattr (fileID, "axes", "two_theta", 9, NX_CHAR);
            NXmakedata (fileID, "two_theta", NX_FLOAT32, 1, &n);
            NXopendata (fileID, "two_theta");
               NXputdata (fileID, tth);
               NXputattr (fileID, "units", "degrees", 7, NX_CHAR);
            NXclosedata (fileID);  /* two_theta */
            NXmakedata (fileID, "counts", NX_FLOAT32, 1, &n);
            NXopendata (fileID, "counts");
               NXputdata (fileID, counts);
            NXclosedata (fileID);  /* counts */
         NXclosegroup (fileID);  /* data */
      NXclosegroup (fileID);  /* Scan */
   NXclose (&fileID);
   return;
}
```

More examples of writing NeXus data files can be found in the *Examples* chapter of the NeXus Reference Documentation.[Page 5, 7]

# 6 More Information

Did we get you interested? Here is where you can get more information. Our main entry point is the NeXus WWW-site at https://www.nexusformat.org/ where you can find more information, download the NeXus API, NeXus User Manual[Page 4, 6] and NeXus Reference Documentation.[Page 5, 7]

If you encounter problems then please help us make NeXus better. Report your problem to the NeXus mailing list (nexus@nexusformat.org). Problems that we never know about have absolutely no chance of getting resolved.

NeXus is a voluntary effort. Thus, if you have spare time and are willing to lend us a hand, you are more welcome to contact us via nexus-committee@nexusformat.org

## 6.1 Who is behind NeXus?

NeXus was developed from three independent proposals from Jonathan Tischler, APS, Przemek Klosowski, NIST and Mark Koennecke, ISIS (now PSI) by an international team of scientists during a series of SoftNess workshops in 1994 - 1996. More work was done during NOBUGS conferences. Since 2001, NeXus is overseen by the NeXus International Advisory Committee (NIAC) which meets once a year. The NIAC strives to have a representative for each participating facility. The NIAC has a constitution which you can find on the NeXus WWW site.