# NeXus data format options for Mantid

## Introduction

The Mantid project is developing a framework for data analysis for neutron and muon data within the STFC ISIS department. This requires the ability to both read existing data files in NeXus format and to be able to write the processed data out in a similar file format. NeXus is a general format for describing X-ray and neutron data which uses either HDF or XML to store the information. Several DTDs have been, or are in the process of being, developed to cater with specific types of instruments. For example there are draft DTDs for Time of Flight (TOF) RAW data, Muon data and processed data. These DTDs build on the available fields defined by NeXus to specify the data which is required or optional for a given area.

# NXprocessed

The NXprocessed DTD represents a basic format that could be used to store some types of Mantid Workspace data. It is intended to represent a set of data that has been processed so that it no longer requires the instrument definition to interpret it. Hence it would not be a suitable format to write a workspace that had just been read in from a RAW file or from the DAE. This is described by the following DTD:

<!--

URL: http://www.nexusformat.org/classes/xml/NXprocessed.xml

Editor: NIAC

NIAC Version: 0.1

$Id$

Template of a generic NXentry containing processed data.

The assumption is that measured data, which could, for example, stored in another

NXentry within the same file, has been reduced to a standard form, e.g., S(Q), so

that the instrument information is no longer required. It is only necessary, therefore,

to store the multidimensional array containing the processed data within one or more

NXdata groups.

This is not a true metaDTD because both the values and the axis names can use

something more descriptive.

-->

<NXentry name="{Name of entry}">

<title>

{Extended title for entry}

</title>

<definition URL="http://www.nexusformat.org/instruments/xml/NXprocessed.xml"

version="1.0">

NXprocessed

</definition>

<NXsample name="{Name of sample}">?

{Any relevant sample information necessary to define the data.}

</NXsample>

<NXdata name="{Name of processed data}">

<values signal="1" type="NX\_FLOAT[:,:]" axes="axis1:axis2">{Processed values}</values>

<axis1 type="NX\_FLOAT[:]">{Values of the first dimension's axis}</axis1>

<axis2 type="NX\_FLOAT[:]">{Values of the second dimension's axis}</axis2>

</NXdata>

<NXprocess name="{Name of process}">?

{Any relevant information about the steps used to process the data.}

</NXprocess>

</NXentry>

The outline is very basic and the details of the NXsample and NXprocess are left open to the requirements of the implementers. Detailed NXsample definitions are available for other DTD’s which are appropriate to the experiments they are describing. A list of standard fields is available on the Nexus website at <http://www.nexusformat.org/NXsample>. The field NXsample can also contain other NeXus fields, such as NXgeometry, NXbeam, NXenvironment, etc. to describe aspects of the sample.

## NXprocess data

If we look at a typical Mantid 2D workspace, generated by LoadRawNexus on the file HET15869.RAW,this currently contains the additional fields for:

* m\_axes (information on the axes including titles)
* m\_title (workspace title)
* m\_comment (comments on workspace)
* sptr\_intrument (shared pointer to instrument description)
  + MantidGeometryCompAssembly
  + \_detectorCache
  + \_sourceCache
  + \_sampleCache
* sptr\_spectramap (shared pointer to spectra to detector map)
  + \_s2dmap (24962 null pointers?)
* sptr\_sample (shared pointer to sample data)
  + m\_name
  + m\_manager
    - m\_properties
    - m\_orderedProperties (sample ordered properties, such as time series values)
      * m\_isDefault (true)
      * m\_name (Log source file name (RAW) or Log name (Muon data(
      * m\_documentation
      * m\_typeinfo
      * m\_direction (0)
* m\_history
  + m\_environment
    - OSname/version/etc
  + m\_algorithm[0]
    - for each applied algorithm name/version/date/duration/properties (set if run as a managed algorithm)
* m\_isdistribution (false)
* m\_noVectors
* data
  + X(),Y(),E()

The LoadRawTest does not populate all possible fields. The Log data is stored in the sample data under m\_manager/m\_orderedProperties.

## Storage in NXprocessed format

To save algorithm history data in the Nexus NXprocessed format, we could use the existing fields that are available there. The NXprocess section of NXprocessed is just defined as containing entries of type NXnote. These notes can take sequence numbers to define the order of the steps. What is not defined is what should be within the note, in terms of the free text field. The NXprocess definition is:

<?xml version="1.0" encoding="UTF-8"?>

<!--

URL: http://www.nexus.anl.gov/classes/xml/NXlog.xml

Editor: NIAC

$Id: NXprocess.xml 4 2005-07-19 04:10:26Z rio $

Template for a process.

-->

<NXprocess name="">

<NXnote name="{numbered name to allow for ordering steps}">

{}{The note will contain information about how the data was processed. The contents of the note can be anything that the processing code can understand, or simple text.}+

</NXnote>

</NXprocess>

The NXnote fields are:

<?xml version="1.0" encoding="UTF-8"?>

<!--

URL: http://www.nexus.anl.gov/classes/xml/NXnote.xml

Editor: NIAC

$Id: NXnote.xml 4 2005-07-19 04:10:26Z rio $

This class can be used to store additional information in a NeXus file e.g.

pictures, movies, audio, additonal text logs

-->

<NXnote name="{name of note}">

<author type="NX\_CHAR">

{Author or creator of note}?

</author>

<date type="ISO8601">

{Date note created/added}?

</date>

<type type="NX\_CHAR">

{Mime content type of note data field e.g. image/jpeg, text/plain, text/html}?

</type>

<file\_name type="NX\_CHAR">

{Name of original file name if note was read from an external source}?

</file\_name>

<description type="NX\_CHAR">

{Title of an image or other details of the note}?

</description>

<data type="NX\_BINARY">

{Binary note data - if text, line terminator is \r\n.}?

</data>

</NXnote>

Of these fields the following might be used to describe the application of a single Mantid algorithm, for example LoadRaw:

<?xml version="1.0" encoding="UTF-8"?>

<!--

URL: http://www.nexus.anl.gov/classes/xml/NXnote.xml

Editor: NIAC

$Id: NXnote.xml 4 2005-07-19 04:10:26Z rio $

This class can be used to store additional information in a NeXus file e.g.

pictures, movies, audio, additonal text logs

-->

<NXnote name="Mantid-Step=1">

<author type="NX\_CHAR">

Mantid/username

</author>

<date type="ISO8601">

20080917T12:00:00

</date>

<type type="NX\_CHAR">

text/plain

</type>

<description type="NX\_CHAR">

Mantid algorithm

</description>

<data type="NX\_BINARY">

Algorithm\_name=”loadraw”

Algorithm\_version=1.0

Algorithm\_parameters=(filename ../../Test/Data/HET15869.RAW)

(workspace outputworkspace)

</data>

</NXnote>

The same format can be used for subsequent algorithmic steps, up to the one that produces the final result. The only new part here is the way the algorithm is described within the data section. Here we have just listed the algorithm name and the required parameters. The remaining three parameters just take their default values. For subsequent steps make sense it is necessary for the workspace names to be consistent across NXnote steps.

An alternative to using the above value pairs would be to write the Python code that could be used to implement the action. However, this would not be useful to usage in C++ and may be limiting.

The environment data, essentially the operating system information, could also be written out here. It may be useful for debugging purposes to know the system that the algorithm was executed on. A separate NXnote could be used for this, assuming that all the processing was done on a single machine. In fact the algorithm can only store one set of environment data, so this seems reasonable.

For example:

<NXnote name="Mantid-environment">

<date type="ISO8601">

20080917T12:00:00

</date>

<type type="NX\_CHAR">

text/plain

</type>

<description type="NX\_CHAR">

Mantid environment

</description>

<data type="NX\_BINARY">

Version=1

Osname=Windows NT

Osversion=5.1

Username=rff93

</data>

</NXnote>

For the case of multiply algorithms applied to one or more data sets the m\_history field contains the sum of histories for each input workspace. For example, if we take the LoadNexus algorithm to create a workspace and then use the Plus algorithm to add two copies of this workspace together, we get three entries in the m\_history/m\_algorithm section, the first two being “LoadNexus” and the third “Plus”. It is not likely that two identical spectra would be added together in practice, though it is possible that a background might be fitted to (parts) of the original data and then subtracted from the original. It is not a problem to have the same entries repeated in the history and is needed to show the data flow.

The algorithm data stored in this simple format is only meaningful within the context of the Mantid framework. While other users could follow the basic flow of the analysis by looking at the names and parameters used within algorithms, they could not reproduce them without having access to the details of the implementation used by Mantid. Other software packages could use a similar format to describe the algorithms they used, but these will also only be fully understood with the package being used.

## NXsample data

The NXsample data is set by what can be read from the input file. In the case of the example EMU files it contains just the name, temperature and magnetic field data, along with a number of time series properties. These can be written to the new data file. If the processed data is stored in the same file as the original data it would make more sense to create a link to the original NXsample section rather than making a copy. It is not clear if the user would want to place additional data within the NXsample section during processing – for example adding unit cell parameters which were not in original NXsample. These might have been determined through processing.

## NXshape

The geometry of objects such as the sample and the detectors is represented by an NXgeometry entry which contains an NXshape field along with translation and orientation information. The NXshape component is currently defined as:

<?xml version="1.0" encoding="UTF-8"?>

<!--

URL: http://www.nexus.anl.gov/classes/xml/NXshape.xml

Editor: NIAC

$Id: NXshape.xml 4 2005-07-19 04:10:26Z rio $

-->

<NXshape name="{name of shape}">

<shape type="NX\_CHAR">

{"nxcylinder", "nxbox", "nxsphere", ...}?

</shape>

<size type="NX\_FLOAT[numobj,nshapepar]" units="meter">

</size>

</NXshape>

The additional notes add: *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.xml class. Physical extent of the object along its local axes (after NXorientation) with the center of mass at the local origin (after NXtranslate). The meaning and location of these axes will vary according to the value of the "shape" variable. nshapepar defines how many parameters. For the "nxcylinder" type the parameters are (diameter,height). For the "nxbox" type the parameters are (length,width,height). For the "nxsphere" type the parameters are (diameter).*

Thus Nexus can deal directly with three of the objects which are currently proposed for the geometry within Mantid, the finite cylinder, the cuboid and the sphere. These could be extended to include the other proposed finite object types, the cone and the hexahedron, e.g. as NXcone and NXhexahedron, with suitable parameters. As the orientation and translation are dealt with separately we only need these objects in a standard alignment e.g. a cone with apex at the origin and axis along the Z axis. The parameters would be height and radius of the base. For the hexahedron, even with one point located at the origin, 7 position vectors are required to locate the points defining the object.

# Mantid Geometry Objects compared to Nexus Shapes

The current suggestion for Mantid geometry objects is based on the Computational Solid Geometry (CSG) models implemented within the Geometry class, based on the MCNPX definitions, with some added options for more convenient definition of common “finite” objects. The MCNP objects are based on surfaces and their intersections and unions to form objects.

The surfaces currently implemented, in general infinite, are:

* Plane
* Sphere
* Cylinder
* Cone
* General quadratic
* Torus (not fully implemented)

These can be used to form finite objects by specifying enclosed volumes, e.g. with 6 planes to define a cuboid. The sphere, certain general quadratic surfaces and torus are special cases where one side of the surface on its own defines a finite volume. The main drawback of a purely CSG description is that fundamental information such as the centre of a cuboid is not trivially extracted from the planes and intersection of them.

The extension proposed for Mantid is to include certain finite *objects* rather than surfaces:

* Finite cylinder – defined by centre point (on base), normal of axis and height.
* Finite cone – defined by point of cone tip, axis vector, angle of cone and height of cone.
* Finite sphere – defined by centre and radius – unlike sphere surface this refers to the volume within the sphere.
* Torus – defined by centre point **C** , normal and two radii, that of the centre of the ring from **C** and of the torus about the ring. Again this finite definition is of the volume within the torus, not of the surface.
* Hexahedron – a volume defined by 8 ordered points giving the vertices of the volume. This is an object that is commonly used in finite element calculations and a collection of these objects can be used to approximate an arbitrary object. Note that in general each face has four points that do need to be in a single plane. If this is the case each face will need to be represented as a quadratic surface in the CSG description, which makes the translation a little more complex than first appears. The additional constraint of the hexahedra is that the edges are linear.
* Cuboid – a right angled hexahedron defined by the four points forming one corner.

Within Nexus the concept of shape is currently much more limited. The NXshape field is currently defined as above to be one of:

* NXcylinder – a finite cylinder that is axis aligned and defined by diameter and height. Presume that height is the z-direction and origin is at (0,0,0).
* NXsphere – a finite sphere located at the origin and of diameter d.
* NXbox – axis aligned cuboid of given length, width and height (x,y,z). Not clear where the origin should be, maybe centre of the cuboid, maybe lower corner.

In each case the object is axis aligned and must be rotated and translated to get the required orientation. The Nexus shape definition is left open in that other objects can be added to it. The TOFRaw definition mentions the need for *spherical slices* that are used to describe area detectors in the case of curved surfaces, without giving details of how they should be defined. The proposed monochromatic reflectometry definition mentions the need for NXslit as a shape, again without a clear description of the geometry.

Thus for a small subset of Mantid objects, the finite sphere, cylinder and box there is a direct Nexus representation of the object, with the addition of the appropriate translation and rotation. I believe that more complex finite objects, such as the spherical shell, can be defined, but there is no standard for the name or the parameter list for such an object.

The translation from a finite NX object such as sphere, cylinder of box to the internal CSG representation is fairly straight-forward. For example the cylinder maps to either directly to the finite sphere proposed above, or to the intersection of two planes with an infinite cylinder. The mapping back from an arbitrary CSG object to these primitives is not possible in general as CSG can represent much more complex objects. Even given a CSG representation of a cuboid, we would still have to check the six planes for orthogonality and then locate corner points to build an NXbox description of the object.

# Storing MCNP objects in Nexus

MCNP allows the construction of complex objects though the use of surfaces which divide space and use of intersection and union operations on the subdivide volumes. Thus a unit cube with lower corner on the origin is described as:

PX 0 ; PX 1 ; PY 0 ; PY 1 ; PZ 0 ; PZ 1

Which defines six planes that we can refer to as 1-6. The object of the cube is then given by the intersection of all these,

+1 -2 +3 -4 +5 -6

MNCP data might be stored in a Nexus file to allow the NXshape field to be extended to include them. For example a surface could be defined as:

<?xml version="1.0" encoding="UTF-8"?>

<!--

-->

<NXmcnpsurface name="{name of surface}" units=meter>

<surface type="NX\_CHAR">

{“surface string”}?

</surface>

</NXmcnpsurface>

This would accept the standard MCNP string definitions such as “PX 0” for the simple x=0 plane up to the general quadratic surface “GQ a b c d e f g h I j k”. MCNP uses integer names to represent surfaces, which then allows the sign operator to define a space from the surface. The current proposal within Mantid is to allow any string name to represent a surface.

These surfaces could then be used to define additional shapes to use within the Nexus NXshape section. For example as:

<?xml version="1.0" encoding="UTF-8"?>

<!--

-->

<NXmcnpshape name="{name of shape}">

<shape type="NX\_CHAR">

{“MCNP object string”}?

</shape>

</NXmcnpsurface>

The string shown above for the intersection of the six plane surfaces to build a unit cube could be used if the surfaces themselves had already been defined. A capped cylinder could be described as:

<NXmcnpsurface name="plane1" units=meter>

<surface type="NX\_CHAR">

{“PZ -2”}

</surface>

</NXmcnpsurface>

<NXmcnpsurface name="plane2" units=meter>

<surface type="NX\_CHAR">

{“PZ 3”}

</surface>

</NXmcnpsurface>

<NXmcnpsurface name="cylinder1" units=meter>

<surface type="NX\_CHAR">

{“C/Z 1 1 0.5”}

</surface>

</NXmcnpsurface>

<NXmcnpshape name="mcnpcappedcyliner1">

<shape type="NX\_CHAR">

{“+plane1 –plane2 –cylinder1”}

</shape>

</NXmcnpsurface>

<NXmcnpsurface name="sphere1" units=meter>

<surface type="NX\_CHAR">

{“SO 1”}

</surface>

</NXmcnpsurface>

<NXmcnpshape name="mcnpcappedcylinerplussphere">

<shape type="NX\_CHAR">

{“+plane1 –plane2 –cylinder1 : -sphere1”}

</shape>

</NXmcnpsurface>

This will defines four surfaces: two Z planes at -2 and +3, an infinite cylinder along the Z direction of radius 0.5 and centered on (1,1,0), and a sphere of radius 1.0 centered on the origin. The object mcnpcappedcylinder1 is built from the intersection of sides of the first three surfaces. Then mcnpcappedcylinderplussphere object is built by the union of the inside of the sphere with the same set of surfaces.

Note that the use of string names in place of integer values will require changes to the existing geometry code. The object names themselves could then be used in addition to the current set allowed in the NXshape field. The mcnp prefix could be used to make the origin of these objects clear.

To maximize the portability of the software it may be best to use the current set of NXshape names if these objects are sufficient and only resort to mcnp definitions when necessary. The additional finite objects that are suggested for Mantid could be mapped to either NXshape types or to mcnp descriptions. The finite sphere, cuboid, cone and cylinder could map to NXsphere, NXbox, NXcone and NXcylinder if they are aligned in the same way as these Nexus shapes. In general they could be mapped as NXgeometry objects with the attached orientation and translation. The hexahedron does not have a Nexus representation. It could be mapped to an mcnp representation for storage in Nexus, with an NXnote field that gives its hexahedral nature.