# 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.