The ISIS NeXus RAW Data file Format

This paper should be considered as "work in progress" and presents a first draft of a new ISIS RAW data file structure, which is based on the NeXus data format [1]. The proposal has been formulated through meetings of the ISIS NeXus Working Group [2] and is based on information from external sources [1,3-7] as well as discussions with members of the ISIS facility. The ISIS Muon community are already making use of NeXus files [7]; this format will aim to encompass their current scheme and be used for Muon RAW data files produced by ISIS DAE-II electronics.

The paper is mostly a collection of group (class) definitions for items that describe an ISIS instrument, the components of an experiment and the data collected. You should not need to be a NeXus expert to understand this paper. The idea is to draw attention to the names and types of information that are proposed for the file, the goal being that any which are missing or not in the correct category (or type/size) to suit a given or proposed experiment/instrument will be spotted. So far work has concentrated on the primary and utility classes; expert input on the instrument component classes (flippers, polarizers etc.) is still required.

Though ISIS is free to use whatever format it wishes for its own RAW files, if this can be matched to an international standard it will assist in data sharing and analysis code portability. A formalisation of the NeXus standard is due to take place in September 2003 [8]

The classes presented here have been based on the current NeXus definitions [1,3], with differences detailed in the text. A tabular form has been used for clarity, but XML DTD versions (as used on the NeXus web site [9]) are available from http://www.isis.rl.ac.uk/computing/NeXus/xml/

For more information and document updates see http://www.isis.rl.ac.uk/computing/NeXus

Please feed any comments and/or additions back to ISIS NeXus Working Group [2] (isis_nexus@isise.rl.ac.uk)

Freddie Akeroyd (Freddie.Akeroyd@rl.ac.uk)
Format Editor, ISIS NeXus Working Group

Contents

Introduction

This document is a draft specification for an ISIS RAW data file within the framework of the NeXus data format [1,3]. ISIS is **committed** to providing standard NeXus data files for its users, but has also **chosen** to use the NeXus format for its RAW data files. NeXus currently provides a convenient set of subroutines for storing data in a portable way, but details such as the names under which these data items are accessed has not been finalised. Recently, a NeXus standards committee [8] was set up with the goal of agreeing version 1.0 of the NeXus data format - the committee is scheduled to meet in September 2003. This is an ideal point for ISIS to consider its future data storage and representation needs and how these might be handled within the NeXus format.

The details and benefits of NeXus, or how files are accessed, will not be covered here – this information is already well documented [1] and is not the aim of the paper. The purpose of this document is to IDENTIFY all the information that one might wish to store in a RAW data file and so enable standard names and locations for them to be assigned. Please note that not all of the information identified here will be relevant for all instruments and/or all experiments; however it is important to identify as much as possible now to minimise the risk of any subsequent addition causing problems. Important things to identify are:

- Is enough information specified to enable a simulation/repeat of the experiment
- Is a quantity specified sensibly and uniquely (e.g. coordinates, units, reference direction)
- Is a quantity always a scalar, or could it be an array in some circumstances
- Could a quantity vary with time/period during an experiment and is this allowed for properly
- How does the definition compare to other standards (e.g. CIF [6]) – can it store the same information

Philosophy

NeXus was devised as an exchange file format i.e. a common standard for sharing data between different establishments; this document details the proposed use of NeXus as a RAW data file format at ISIS and so addresses some different issues. Though this scheme has been devised with ISIS in mind, it should be general enough to handle most data and its creation throws up some interesting issues and discussion points.

RAW data file are archived and provide a historical record of work at the facility and the performance of instruments – from an “exchange file” point of view, much of this sort of information is irrelevant. RAW data files may also contain extra monitoring information – the experiment could have run automatically overnight and control parameters might have wandered more than planned, which would need to be checked during analysis. RAW data files also need to store information on all the individual detectors so any that are later found to be “noisy” can be excluded from analysis; in an exchange file the data may have undergone a first round of detector grouping and/or data reduction. The efficient representation of the detector configuration is thus much more important for the RAW data file.

At ISIS all instruments currently run a common data acquisition and control system, use a common RAW data file format and can use the same general program (GENIE) for first-line data analysis. To allow this to continue, the new structure must be flexible enough to store data from any ISIS instrument in a way that a generic program can interpret. Achieving this adds some complications, such as indexing data against indirect quantities (“spectrum” and “period”) to allow arbitrary parameter variation, but provides a very general mechanism.

NeXus Structure

Think of a NeXus file as like a “file system” and you will not be far wrong. A NeXus file consists of a set of “folders”(groups) containing “information” - each folder (group) has an associated “class”, which determines the names and contents of files (and other folders) that can appear within it. Like a file system, NeXus has a hierarchy and some classes must be present in certain location. Also, like a file system, items (files or folders) can be linked (different locations pointing at the same data).

NeXus defines the following basic data types:

NeXus Type	**Typical C Equivalent**	**FORTRAN Equivalent**	**Description**
NX_INT8 / NX_UINT8	char / unsigned char	INTEGER*1	signed / unsigned 8 bit integer
NX_INT16 / NX_UINT16	short / unsigned short	INTEGER*2	signed / unsigned 32 bit integer
NX_INT32 / NX_UINT32	long / unsigned long	INTEGER*4	signed / unsigned 32 bit integer
NX_FLOAT32	float	REAL*4	32 bit floating point
NX_FLOAT64	double	REAL*8	64 bit floating point
NX_CHAR	char	CHARACTER	8 bit character

All of these basic types can be declared as arrays; strings are handled as one dimensional NX_CHAR arrays. The use of “Typical” for the “C” declaration stems from C only defining a minimum size for a basic type: “long” must be at least 32 bits, is exactly 32 bits on most computers, but is 64 bits on computers running the HP Tru64 operating system.

NeXus is pseudo object oriented – it supports classes, but not inheritance. In a fully object oriented system you could have a general instrument class and then derive/subclass special instances of e.g. powder diffractometer. When a program examines a file and asks for “NXinstrument”, it would be automatically given “NXpowderdiffractomer” as this is a subclass of “NXinstrument”. Unfortunately, this feature is not available in HDF [10] upon which NeXus is based. Maybe it would be useful is NeXus considered adding this – possibly via a “parent” or “superclass” attribute?

A NeXus file contains three sets of quantities: a description of the instrument, the data collected during the experiment and additional information on things like sample environment and experimental procedure. Ideally NeXus classes should correspond to “real world” object and their use aid with structure and understanding. NeXus already defines several classes (e.g. NXsample, NXchopper), but it may be sensible to split things further. For example, the NXmoderator class was not in the original specification and details for the moderator were instead stored as part of the “source” (NXsource).

The layout of the instrument should be specified in the single instance of the NXinstrument class, which will contain various members (NXchopper, NXdetector, NXmoderator etc.) whose spatial locations are specified by their NXposition members. The instrument is broken down into “components” that can be assembled to re-generate the experimental setup – ideally there should be enough information in the file to allow a simulation to be performed of the experiment. The NXsample class will contain information about the entity under investigation during an experiment, with details of the sample environment equipment used contained within instances of NXenvironment.

Mandatory and Optional Items

Not everything mentioned here will be present in every NeXus file - the programming interface does not make any restrictions on what can be put into a file or how it should be named. The current plan is to just name all that might be useful - in another iteration of the standard, what is optional and what is mandatory for a given type of instrument will need to be defined; otherwise it will be impossible to check a NeXus file as “conforming to the standard” except at the most basic level. Currently an XML DTD or schema [9] is envisioned to specify the file contents and to validate against – the name of the DTD/XML schema used will be attached to the appropriate NXentry in the “analysis” variable.

Differences from Current Published NeXus Format

The following are notable – for others see under the individual class descriptions:

- Three new data type notations have been introduced – they are implemented as standard existing data types, but signify the contents should be interpreted in a specific way
  - NX_BINARY: An array of unsigned 8 bit integers (NX_UINT8), but containing arbitrary binary data and not plottable numbers e.g. an image in the NXnote class
  - NX_BOOLEAN: An NX_INT32, but only ever set to 0/1 for False/True
  - NX_TEXT: An NX_CHAR array, but with a specific convention for indicating “end of line”, “timestamp” and “next entry”; for e.g. simple log messages – see below
- The concept of “periods” and the use of the “scanned” attribute (see below)

- The “distribution” attribute (on NXdata to differentiate “counts” and “counts per units axis”)
- NXposition, NXdistance and NXorientation classes for specifying and linking instrument component positions
- NXnote and NXnotebook classes for messages and other data (e.g. images)
- NXdae (optional) for facility specific details of the Data Acquisition Electronics and/or details of the current running acquisition system
- Introducing the idea of “components” to NXsample and creation of NXenvironment and NXsensor classes
- Introducing the idea of “spectra” to NXdetector and enabling efficient storage of array detectors

Conventions Used in this Document

The Name column in a table identifies an item in an instance of a NeXus class. Items can have extra “meta data” associated with them, which are called attributes – these, if any, are listed in the next few lines in the attributes column. Any variables in the attributes column are always attached to the previous variable in the Name column above them; if the Name of the variable is the same as the class (e.g. NXfile), then the attributes are associated with an instance of that class (global) and not any of its members.

Identifying Mandatory and Optional Components

The following convention will be used:

- Variables in **bold** in the Name column of tables are mandatory – they must be present in ALL NeXus files; otherwise they are optional and their inclusion will depend on the instrument, experiment or presence of other items in the class (see the class description of usage)
- Variables in *{italics}* in the Name column are examples of names and any variable name can in fact be used; variable names in normal type mean that exact name must be used

This information is also included in a RE column (the name derives from the fact that a “Regular Expression” is used in the XML DTD [9]). Thus:

Font/style in Name Column	RE Column	Meaning	XML DTD [9]
Something	0/1	A single instance of this variable may be present (optional) – if it is, it must be called “something”	?
Something	1	A single instance of this variable must be present (mandatory) and called “something”	
{Something}	0+	Zero or more variables of this type/class may be present (optional) and can have any unique name(s)	*
{Something}	1+	One or more variables of this type/class must be present (mandatory), but can have any name(s)	+

The above convention dictates that the name for any item that occurs only 0 or 1 times is fixed; this is not required by the current NeXus standard, but would add clarity and ease of location if implemented.

Date and Time

Date and time are stored in ISO8601 format [12] (e.g. 1996-07-31T21:15:22+0600). This time format only allows for ABSOLUTE time, so for delta (relative) time another scheme is needed (see NXlog). The +0600 refers to the time zone, with +Z meaning UTC. Sub second times are supported by specifying “.xxx” after the seconds.

Units

Unit names should be specified in the singular (“second” rather than “seconds”) as per UDUNITS [11]. All physical quantities should have the “units” attribute (NX_CHAR data type) set to the appropriate value. It is intended that the NeXus interface will support unit conversions, but until this is available it is recommended that values are stored in the preferred unit (see the description section of each individual units entry).

Storage of Text and Simple LOG Information (NX_TEXT)

Lines of text should be stored in a one dimensional NX_CHAR array with the line terminator “\r\n” i.e. <carriage return><line feed>; to break a section of text from another, use the form feed character “\f” followed by "\r\n". A time stamp, if any, for the entry should be placed on a separate line before the text with the word "TIMESTAMP" (in capitals) in front e.g.

```
TIMESTAMP:1996-07-21T21:55:22+0600\r\n
My message line 1\r\n
My message line 2\r\n
\f
```

Coordinates/Positions/Orientations/Distances
The instrument is set up in a "global" coordinate system with the MCSTAS convention of:

- Z axis points in the direction of the incident beam
- X axis is perpendicular to the incident beam in the horizontal plane, pointing left as seen from the source
- Y axis points upwards perpendicular to the beam in the vertical plane

The origin of coordinates is arbitrary, but all components in the file must either agree on its absolute location or use relative positioning. One choice of origin is the sample position, but on instruments with very large moving samples this is not so useful. An alternative choice is the "scattering centre", the point in space at which all the detectors are focussed. One advantage of the "scattering centre" is that the spherical polar coordinate specifications of the detector positions are then conveniently related to scattering angles and lengths for direct geometry instruments. To allow for generality, an **origin** member has been defined in NXentry; its use will be detailed shortly.

Individual components of the instrument (e.g. jaws) will have their own set of local axes (x,y,z) which will be fixed to their body in a way defined by their shape. These local axes will probably not coincide with the global instrument axes and so a set of rotation angles will also need to be stored. For this an NXposition class is defined, along with NXdistance and NXorientation; the hope is to provide a general enough method for relating the location of any object with respect to another object. The mechanism also allows for specifying one position relative to another component: a NeXus file link is made in one instance of an NXposition object to another NXposition object and a program can then traverse the chain of links to calculate an absolute position.

NeXus does not need to define absolutely where to place the "origin". All components can instead be declared with a relative position that ultimately follows a chain back to one object; this will be named "origin1", be of class NXposition and a member of NXentry. The real space location of this origin is chosen for convenience and should be mentioned in the description attached to "origin1". If the origin is taken at the sample, then "sample.position.distance" will always be (0,0,0) relative to "origin1"; if the origin is taken elsewhere this will not be so, but everything will still work. It may be convenient to define extra origins (similar to "arms" in MCSTAS) at other parts of the instrument. For example, defining one at the centre of a circular array of detectors would allow their positions to be conveniently specified in spherical polar coordinates. Another possibility would be to define the sample relative to "origin1" and the detectors to "origin2"; the detectors could then be rotated by a rotation of "origin2" without modifying NXdetector.

As well as specifying the component location, it is also necessary to specify the beam direction. Unless otherwise given in an **NXbeam** member of the component, the incident beam is assumed to be travelling along (0,0,+z) in the coordinate system of the object (or origin) our position was defined relative to. Thus, for a component with absolute positioning the beam will always be in the incident beam direction unless specified by an NXbeam member.

The above mechanism may seem overcomplicated and is not definitely decided upon. For example, is the option to specify relative instrument component positions really needed? It was included so a NeXus file might be used as input to a simulation program where relative positions are a convenient way to specify the setup. An alternative to an NXposition class would be e.g. separate three element distance and orientation arrays in each NeXus class which needed them (or even just NX_FLOAT32 position[7] containing: type ("Cartesian", "spherical"), 3 distances and 3 angles). The advantage of an NXposition would come when NeXus was hooked up to an object based scripting language: as the numbers would then be associated with the NXposition class, operator overloading could be used to specify how positions would "add" and "subtract" etc. and methods defined for "NXposition" objects to provide easy conversions between coordinate frames.

Size and Shape

Many instrument components define "height" and "width" variables to specify their size when rectangular, a "radius" variable for when circular etc. Rather than all these different names, an alternative scheme is proposed based on the "shape" of the object and the local coordinate axes this shape defines. All object would just need to specify a **shape** ("cuboid", "cylinder" etc.) and a size array. Specifying **size[3]** would give the dimensions of the object along its local (±x,±y,±z) axes; specifying **size[6]** would give the extent along (+x,+y,+z,-x,-y,-z) and allow for e.g. asymmetric jaws where the reference point may not be the centre of the rectangle. For example take shape="cylinder": the NXdistance variable of **position** would define the location of the reference point for the origin of the local axes: z in the direction of the cylinder axis, x and y in plane. With no rotation the object would be oriented with its local axes pointing in the direction of axes of the object it was defined relative to, but this can be altered with the NXorientation variable within position. If a **size[3]** array variable was specified, the reference point must be the centre of the cylinder and the dimension are size[0]=size[1]=radius, size[2]=length/2). If **size[6]** was specified then the reference point would be elsewhere in the object, with its distance from the cylinder edges along the various axes given by elements of the size[6] array. See NXsample for an example of usage.

Special Array Size variables

The following variables are used in tables:

np	The array is of a size equal to the number of periods in the run
ntc	Number of time channels on the NXdetector (the will thus be ntc+1 boundary values for the histogram)
ns	Number of spectra on the NXdetector
nd	Number of detector elements on the NXdetector
nda	Number of 1D PSD detectors in a 2D array in NXdetector
n_comp	Number of components to the NXsample

Enumerated Strings

An enumerated string is a string variable (NX_CHAR array) whose contents should only be one of a list of specified values. In the description section this is represented as:

"string1" | "string2" | "string3"	The value, if present, must be either "string1", "string2" or "string3" etc.

NXlog Variable Names

Names of form "*_log" are always entries of type NXlog – they show the time dependence of the corresponding quantity "*". For example, the temperature variable would store the average temperature and the time dependence, if specified, would be stored in the variable temperature_log of type NXlog

Case Sensitivity

Variable names in HDF files are case sensitive and so NeXus has chosen to use lower case names of variables wherever possible to avoid confusion.

Data compression

Arrays can be individually stored in compressed format – an option just needs to be selected when the data is written. As compression can be time consuming, it is likely that the ISIS RAW file will be initially written out uncompressed and then converted to a compressed format offline. Compression can also solve the problem of data duplication – storing a large array containing the same number takes up little space with "run length encoding" and so allows a general case to be considered without a storage penalty for simple cases.

Array Ordering Convention

The C language convention is used in this document, which is "fastest varying array index last". Thus the entry a[np,3] in a table is equivalent to the C declaration a[np][3] and corresponds to np blocks of 3 items arranged sequentially. If you were to read such an item into a FORTRAN program, you would declare the same array as A(3,NP) - FORTRAN indexes arrays "column wise" rather than "row wise". This convention is purely typographic and does not have any bearing on the efficiency of using the array in either language.

Type Fields

Often a "type" field is needed in a NeXus class – having this as NX_CHAR and using an "enumerated string" is descriptive, but it has the disadvantage that a reading program has to do string comparisons to determine a course of action. Using an integer number to represent each possible case is quicker for an analysing program and less error prone (no misspelling or case issues), but is less descriptive (you need to examine a separate document to determine what the number means). Maybe the answer is to store both in separate "type_name" and "type_code" fields?

Standard Variable Names

While a program can search for instances of a given class, it is easier if standard names are used for certain common quantities. Suggestions are:

- A variable named *_env is always of type NXenvironment and is related to a variable called *
- A variable named *_log is always of type NXlog and related to a variable called *
- Any extra information on a class is stored in a member called either notes or notebook
    - The notes class member is always of type NX_TEXT
    - The notebook class member is always of type NXnotebook
- The description of a class is always stored in the description member of type NX_CHAR
- The spatial location of an object is always called position and of type NXposition
- Information about the (neutron) beam is contained in the beam variable of class NXbeam
- The name of an NXdata variable is the same as the NXmonitor or NXdetector instance it refers to
- Coordinate origins should always be of class NXposition and named origin1, origin2 etc.

NeXus API Extensions

It would be useful to know where a NeXus link is to, and what is linked to a given object. Would it be possible for this for be written into "forward_link" and "backward_link" attributes automatically by the interface?

Periods and the SCANNED attribute

At ISIS the memory of the Data Acquisition Electronics can be blocked into sections call "periods". At a given time, all detector output will be going into a specific period – changing a period is similar to starting a run except:

- It is a very quick operation as no data is copied to the host computer (only a memory pointer is moved in the electronics)
- Periods can be triggered by an external signal source to the electronics on a "frame by frame" basis as well as from the control computer
- You can go back and resume counting in a previous period as data memory is not cleared

Periods are often used for cycling round a set of experimental conditions – detector positions as well as sample environment may change. To describe this, the SCANNED attribute has been defined. If this attribute is present and set to TRUE, the quantity concerned will be an array with the "slowest varying dimension" of size [np], the number of periods. In accordance with the array convention mentioned above, [np] will always be written first in the array dimensions list.
Though a period is similar to having an extra dimension to the raw data of e.g. "number of temperature steps", it is far more general - many quantities (pressure, temperature, detector position etc.) may be varied simultaneously.

Connecting Detectors and Monitors with Data

A simple scheme has been chosen where the name used for the detector bank (class NXdetector) or monitor (class NXmonitor) is also used for the corresponding NXdata class instance containing the counts. There is no name clash as the NXdetector/NXmonitor instance is "one level down", hidden in the NXinstrument class.

Primary Classes

NXfile

NXfile is not a real class in the NeXus file – it is just a convenient name under which to group the global attributes (properties/variables) of the NeXus file (the class is also referred to as NXroot)

RE	Name	Attribute	Type	Value	Description
	NXfile			Top level class	
1		file_name	NX_CHAR	File name of original NeXus file	To assist in identification if the external name has been changed
1		**file_time**	ISO 8601	Date and time of file creation	
1		**file_update_time**	ISO 8601	Date and time of last file update/modification/change	
1		**nexus_version**	NX_CHAR	Version of NeXus programs (API) used in writing the file	
1		**hdf_version**	NX_CHAR	Version of NCSA HDF library used by NeXus to create file	
1		**creator**	NX_CHAR	Name of user producing the file	This is different to user doing the experiment (NXuser)
0/1		affiliation	NX_CHAR	Affiliation of creator	
0/1		address	NX_CHAR	Postal address (complete) of creator	
0/1		telephone_number	NX_CHAR	Telephone number of creator	international format
0/1		fax_number	NX_CHAR	Fax number of creator	
0/1		email	NX_CHAR	E-mail address of creator	
1		**file_changes**	NX_TEXT	Brief log of changes to the file	Automatically updated by NeXus interface
1		**checksum**	NX_CHAR	checksum	checksum computed from all data arrays; may be digitally "signed"
0/1		checksum_type	NX_CHAR	checksum algorithm used	Assume MD5 if not present
0/1		signature	NX_CHAR	Pointer to digital signature certificate	Its presence indicates the checksum has been signed
0/1		signature_type	NX_CHAR	Checksum signing method	e.g. PGP
1		**unique_id**	NX_CHAR	UUID identifying this file uniquely	Automatically changed if file is modified
1+	***{entry1}***		NXentry	First entry	First item in file will be the raw data; subsequent entries may be analysed data etc.

Differences from Current NeXus Standard
The **user** attribute has been renamed **creator** – this is what it really refers to; the "user" of the instrument (experimenter) is specified in the NXuser class within NXentry. Also new are:

file_update_time, **file_changes**, **cksum**, **cksum_type**, **signature**, **signature_type**, and **unique_id**.
It would be useful to know the last time the file was written to as well as when it was created; this can be determined by the interface and inserted automatically into **file_update_time**. Also the interface could automatically keep a brief log of updates to the file in the **change_log** variable – things like the date classes were added or modifed (user supplied comments would be added to the NXentry **notebook** variable instead)
Data integrity and validation have not been discussed so far. Though a full mechanism is not presented, producing a checksum of all data in the file and then optionally signing this would cover most eventualities – encryption could be done externally or maybe HDF will support it?

Notes
The NXentry class is the only class allowed at the top level of the file and there must be at least one instance of it. The DTD specifying the NXentry definition is contained within it, but is one needed for NXfile as well? As most of the global variables are added by the interface maybe this is linked to nexus_version attribute?

NXentry

This is the top level NeXus group and contains a complete set of measurements (a "run" at ISIS). Conventionally separate instances of NXentry are named "entry1", "entry2" etc. It is mandatory that there is at least one group of this type in the NeXus file.

RE	Name	Attribute	Type	Value	Description
	NXentry			name of entry	
0/1	type		NX_CHAR	"raw" | "processed"	type of information stored in entry
1	**title**		NX_CHAR	Main title for the whole entry	
1	**analysis**		NX_CHAR	Analysis name	This specifies the template the entry was based on i.e. the name of the definition file giving mandatory and optional fields.
1		**URL**	NX_CHAR	http://some.where	URL of XML DTD or schema
1		**Version**	NX_CHAR	$Revision: $	XML DTD version e.g. 1.0.0; inserted automatically by CVS
1	**start_time**		ISO8601	Start time of entire measurement	
1	**end_time**		ISO8601	End time of entire measurement	
1	**duration**		NX_FLOAT32	Duration of data collection measurement	This will not be "start-end" as data may not be collected at all times due to e.g. a temperature going out of range
1		**Units**	NX_CHAR	second	
1	**run**		NXrun	details of the run	
1	**run_number**		NX_INT32	Unique identification number of run/scan stored in this entry	
1	**run_cycle**		NX_CHAR	The current scheduled machine operation period	
0/1	program_name		NX_CHAR	Name of program used to generate file	
1		**version**	NX_CHAR	Generating Program version number	
0/1	command_line		NX_CHAR	Contents of any command line used to generate file	

1	**notebook**		NXnotebook	Log of useful stuff (history) about the experiment supplied by the user	
1+	***{user1}***		NXuser	Details of users involved with the experiment	
1	**sample**		NXsample	sample	Details of the sample under investigation
1	**instrument**		NXinstrument	instrument name	Details of the instrument used
1+	***{bank1}***		NXdata	data	The data collected; this is named after the corresponding NXdetector in the NXinstrument group
1	**program_notes**		NX_TEXT		Log from instrument control program (dates of period changes, CAMAC waiting etc). begins, ends, pause, resume etc
1	**experiment_identifier**		NX_CHAR	Experiment identifier/number	For ISIS, the RB/proposal number of the experiment
1+	***{origin1}***		NXposition	Origins for relative component placement	origin1 is the global instrument reference point

Differences from Current NeXus Standard
New variables: **type, run_cycle, program_notes, experiment_identifier, run (**class **NXrun), origin (**class **NXposition)** and **notebook (NXnotebook).** The NXnotebook class is detailed later and is a collection of **NXnote** entries for the run. The NXmonitor instance has been moved to NXdetector and there can be multiple instances of NXuser.
The global instrument origin is indicated by the member **origin1;** all position can be done relative to this. Additional origins can also be created if they are useful – they should be named origin2, origin3 etc.
Would it be more logical to rename **analysis** as **template**?

Notes
NXdata members are named the same as their corresponding NXdetector and NXmonitor members in NXinstrument

ISIS Notes
A place is needed for the contents of DAE-II focussing memory

NXrun

This class contains information about the experiment that has been separated from NXentry for convenience.

RE	Name	Attribute	Type	Value	Description
	NXrun			Name of run	
1	**period_start_time**		ISO8601[np]	First time each period started	"First" needed as periods may be cycled
1		**Scanned**	NX_BOOLEAN	0 | 1	
1	**period_end_time**		ISO8601[np]	Last time each period ended	"Last" needed as periods may be cycled
1		**Scanned**	NX_BOOLEAN	0 | 1	
1	**period_duration**		NX_FLOAT32[np]	Time spent in the period	Summed over all period cycles
1		**Units**	NX_CHAR	second	
1		**Scanned**	NX_BOOLEAN	0|1	
1	**period_cycles**		NX_INT32[np]	Number of times data collection took place in each period	If zero, space was allocated for the period but it was not used; if greater than one, periods were cycled (repeated)
0/1	short_title		NX_CHAR[np]	A per period title	
1		**scanned**	NX_BOOLEAN	0 | 1	
1	**total_charge**		NX_FLOAT32		Total charge (integrated beam current) in all periods
1		**Units**	NX_CHAR	Micro.amp.hour	
1	**total_raw_frames**		NX_INT32		Total number in all periods
1	**total_good_frames**		NX_INT32		Total number in all periods
1	**charge**		NX_FLOAT32[np]		Charge (integrated beam current) in each period
1		**Scanned**	NX_BOOLEAN	0 | 1	
1		**Units**	NX_CHAR	Micro.amp.hour	
1	**raw_frames**		NX_INT32[np]		Raw frames for each period
1		**Scanned**	NX_BOOLEAN	0 | 1	
1	**good_frames**		NX_INT32[np]		Good frames for each period
1		**Scanned**	NX_BOOLEAN	0 | 1	

Differences from Current NeXus Standard
This class does not exist in the current NeXus standard and most of its members are new; the use of periods is covered in the introduction.

Notes
The **period_start_time** variable is only updated once data collection has actually began

NXuser

Definition of experiment user contact information

RE	Name	Attribute	Type	Value	Description
	NXuser			Name of user	
0/1	Local_contact		NX_BOOLEAN	0 | 1	TRUE is the user is an ISIS staff member assigned to assist with the experiment
0/1	Primary_user		NX_BOOLEAN	0 | 1	TRUE if the primary user/investigator (e.g. experiment proposer)
1	**Name**		NX_CHAR	Full name of user	Surname, first name(s) e.g. Other, A. N.
0/1	Affiliation		NX_CHAR	Institute	
0/1	Address		NX_CHAR	Full postal address of user	
0/1	Telephone_number		NX_CHAR	Telephone numbr in international format	e.g. +441234567890 use ; to separate multiple numbers if required
0/1	Fax_number		NX_CHAR	Fax number in international format	e.g. +441234567890 use ; to separate multiple numbers if required
0/1	Email		NX_CHAR	Email address	e.g. a.n.other@rl.ac.uk
1	**User_identifier**		NX_CHAR	Unique facility based identifier for this user	User number at ISIS
0/1	role		NX_CHAR	role of user	e.g. “co-investigator”

Differences from Current NeXus Standard
primary_user, role, user_identifier and **local_contact** are new; there can be multiple instances of NXuser

Notes
The suggestion of **primary_user** is taken from Cooper *et al*. [4]; there is only one primary user. Though information in NXuser may quickly become out of date (people move, phone numbers and email addresses change) it provides a useful historic record when the file is archived. If the “user_identifier” field is present, it could be used to locate the current address of the user in the facility’s user database. The NXuser class could also be used in subsequent NXentry instances to record information about the person who is analysing the data stored in the entry.

NXdata

Definition of plottable data and their dimension scales. It is mandatory that there is at least one group of this type in each NXentry group. The "signal" and "axes" attribute of the "counts" item define which items are plottable data and which are dimension scales.

RE	Name	Attribute	Type	Value	Description
	NXdata			Name of data	
0/1		calibration_status	NX_CHAR	“nominal” | “measured”	
1	**counts**		NX_INT32[np,ns,ntc]	data values	
1		**Signal**	NX_CHAR	1	Identifies the main plottable array
1		**Axes**	NX_CHAR	“[time_of_flight,spectrum_index,period]”	
1		**Long_name**	NX_CHAR	“neutron counts”	
1		**units**	NX_CHAR	“counts”	
1		**Checksum**	NX_INT32		
1	**time_of_flight**		NX_FLOAT32[ntc+1]	time channel bin boundaries	linked to NXdetector variable; need NX_FLOAT64?
1		**Long_name**	NX_CHAR	“time_of_flight”	
1		**Units**	NX_CHAR	Micro.second	Might want to use clock pulses instead for accuracy
1		**Axis**	NX_INT32	1	1 = fastest varying array index
1		**Distribution**	NX_BOOLEAN	0	pure counts
0/1		first_good_bin	NX_INT32	location of first bin with meaningful data	
0/1		last_good_bin	NX_INT32	location of last bin with meaningful data	
0/1		t0_bin	NX_INT32	location of “time zero” bin	
1	**spectrum_index**		NX_INT32[ns]	global spectrum number:	linked to NXdetector variable
1		**Long_name**	NX_CHAR	“spectrum_index”	
1		**Units**	NX_CHAR	none	
1		**Axis**	NX_INT32	2	
1		**Distribution**	NX_BOOLEAN	0	pure counts
1	**period**		NX_INT32[np]	period number	
1		**Long_name**	NX_CHAR	“period_number”	
1		**Axis**	NX_INT32	3	
1		**units**	NX_CHAR	none	
1		**Distribution**	NX_BOOLEAN	0	pure counts
1	**title**		NX_CHAR	Title for data/plot	

Differences from Current Standard

The **distribution** attribute is new. If this is 1 (true) the counts are per unit axis; if 0 (false) just pure counts; if not present, assume true (i.e. counts/units axis). Why is it needed? The units field alone is not enough to help us. For example the counts we may have normalised to the total duration of the experiment (time) and so the units will be counts / time, but this time is not "time_of_flight".

At ISIS an errors array is not stored with RAW data as this can be calculated by sqrt(counts). The current NXdata standard only allows for one errors array – we propose that the errors array should be called "*_**errors**" and refer to variable "*****". An alternative would be to have an attribute "errors" on the major variable giving the name of the errors arrays, in the same way as the "axes" variable does for dimension scales.

first_good_bin and **last_good_bin** indicate which part of the data range is meaningful – counting may have started before real data had arrived. The **t0_bin** indicates the centre of the Muon pulse [7]

Notes

The histogram_offset variable is not set as the data is being stored as "real histograms" and not "bin centres". The "Axis" attribute is the old method of indicating dimension scales – it will be written for ISIS Muon backward compatibility.

ISIS Note

Need to decide where spectrum 0 (unassigned detector output) and time channel 0 (data collected before timing has started) will go. spectrum 0 could go to NXdae and time channel 0 be stored as normal in NXdata, but with **first_good_bin** set accordingly.

NXlog

Definition of logged information, i.e. information monitored during the run. They contain the logged values and the times at which they were measured in either ISO 8601 format, or as elapsed time since the beginning of the run. 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.

RE	Name	Attribute	Type	Value	Description
	NXlog			Name of log	
0/1	Time		ISO8601[i]	Time of logged entry	
0/1	delta_time		NX_FLOAT32[i]	Relative time of logged entry	
0/1		units	NX_CHAR	second	
0/1	delta_offset		ISO8601	Origin of delta_time	Probably a link to the run start time in NXentry
1	**Value**		NX_FLOAT32[i,n]	value of logged variable	
1		**Units**	NX_CHAR		
0/1	raw_value		NX_FLOAT32[i,n]	raw value of logged variable	e.g. millivolts from a thermocouple
0/1		Units	NX_CHAR		
1	**Description**		NX_CHAR	Description of measured quantity	

Differences from Current NeXus Standard
ISO8601 only allows absolute time so to specify relative time (e.g. from the start of run) we have proposed the **delta_time** and **delta_offset** members An alternative would be to still use the ISO8601 string, but define a leading "+" character to mean "relative time"
The original NXlog definition has temperature, electric_field etc as separate entries in the NXlog. We also feel [4] this is too restrictive on logging times and prefer a separate NXlog per variable (**value**); in addition the **raw_value** from a sensor can be recorded. All variables of type NXlog should be named "***_log**" so they can be tied up to a nominal quantity "*" if it is present. The **description** field is also new.
The **value** array has been made multidimensional to allow vector (n=3) or tensor (n=6) quantities to be stored. The order for storing symmetrical tensor values (such as stress) needs to be specified e.g. T[i,j] with {i,j}={1,1}{2,2}{3,3}{2,1}{3,2}{3,1}

Notes
There could be a lot of logging information, but it all needs to go somewhere and the NeXus file is as good a place as any. If a value is repeated or changes very little, compression should help.

NXsample

Definition of the sample under investigation. With very few exceptions, samples are usually measured whilst inside a container inside the chosen sample environment, and the scattering or absorption from the container must be corrected for in the final analysis. It could be argued that the container is part of the sample; it could, just as easily, be argued that the container is part of the sample environment apparatus. Either way, we must allow for this somehow.
To complicate matters further, many samples are multi-component (e.g. metal alloys, a polymer in a solvent, a polymer blend, an oil-in-water emulsion stabilised by adsorbed surfactant, water condensed in Vycor, a magnetic multilayer, adsorbates on a catalyst, etc.). A single sample formula, or scattering cross-section, or unit cell, etc., is inadequate to describe the actual sample being investigated. To overcome this, we define most elements as arrays of size **n_comp** and treat the sample can as part of the sample. An extra array “**sample_component**” indicates whether a component is “of interest” or “part of the kit” (and hence allowed for later in a calculation)

RE	Name	Attribute	Type	Value	Description
	NXsample			Name of sample	
1	**name**		NX_CHAR	Sample identification code	
1	**type**		NX_CHAR	“sample” | “sample+can” | “can” | “calibration sample” | “normalisation sample” | “simulated data” | “none” | “sample environment”	
1	**situation**		NX_CHAR	“air” | “vacuum” | “inert atmosphere” | “oxidising atmosphere” | “reducing atmosphere” | “sealed can” | “other”	The “atmosphere” will be one of the components, which is where its details will be stored; the relevant component will be indicated by the entry in the sample_component member
0/1	changer_position		NX_CHAR	Position on sample changer	Was NX_INT32, but NX_CHAR is more general
1	**description**		NX_CHAR	Description of the sample	
0/1	preparation_date		ISO8601	Date of preparation of the sample	
1	**shape**		NX_CHAR	“sphere” | ”shell” | ”cuboid” | ”cylinder” | ”tube”| ”single crystal” | “general”	general will require more thought
1	**position**		NXposition	The position and orientation of the reference point (probably centre) of the sample	The meaning of centre will be defined by the sample shape
0/1		scanned	NX_BOOLEAN	0 | 1	The sample may move/rotate
0/1	beam		NXbeam	Details of beam incident on sample	used to calculate sample/beam interaction point
1	**Component**		NX_CHAR[n_comp]	Details of the component of the sample and/or can	

1	**Sample_component**		NX_CHAR[n_comp]	“sample” | “can” | “atmosphere” | “kit”	Or use an NX_INT32 instead?
0/1	Chemical_formula		NX_CHAR[n_comp]	Empirical chemical formula of each component	
0/1	Molecular_weight		NX_FLOAT32[n_comp]	Molecular weight (C=12) of each component	
0/1	Concentration		NX_FLOAT32[n_comp]	Concentration of each component	
1		**units**	NX_CHAR	g.cm-3	
0/1	Volume_fraction		NX_FLOAT32[n_comp]	Volume fraction of each component	
0/1	Density		NX_FLOAT32[n_comp]	Density of each component (g cm^{-3})	
0/1	Scattering_length_density		NX_FLOAT32[n_comp]	Scattering length density of each component (cm^{-2})	
0/1	Coherent_cross_section		NX_FLOAT32[n_comp]	Coherent cross section of each component (fm)	
0/1	Incoherent_cross_section		NX_FLOAT32[n_comp]	Incoherent cross section of each component (fm)	
0/1	Absorption_cross_section		NX_FLOAT32[n_comp]	Absorption cross-section of each component (fm)	
0/1	Unit_cell		NX_FLOAT32[n_comp,6]	Crystallographic a/b/c/alpha/beta/gamma	
0/1	unit_cell_class		NX_CHAR[n_comp]	"cubic" | "tetragonal" | "orthorhombic" | "monoclinic" | "triclinic"	Not very descriptive, but may be the only information available
0/1	Unit_cell_volume		NX_FLOAT32[n_comp]	Volume (nm^3) of the unit cell of each component	
0/1	unit_cell_group		NX_CHAR[n_comp]	Crystallographic point or space group	
0/1	Mass		NX_FLOAT32	Mass of sample (g)	
0/1	size		NX_FLOAT32[3]	Size of sample along its local “x”, “y” and “z” axes	The sample axes directions are defined by its shape and rotate with it
0/1		units	NX_CHAR	mm	
0/1	inner_size		NX_FLOAT32[3]	Inner dimensions of the sample along its local “x”, “y” and “z” axes if it is hollow	The sample axes are defined by its shape
0/1		units	NX_CHAR	mm	
0/1	path_length		NX_FLOAT32	Path length through sample/can (mm)	For simple case when it does not vary with scattering direction

0/1	path_length_window		NX_FLOAT32	Thickness of a beam entry/exit window on the can (mm)	assumed same for entry and exit windows
0/1	orientation_matrix		NX_FLOAT32[n_comp,3,3]	Orientation/UB matrix of a crystalline sample	See [13] for definition
0/1	Transmission		NXdata	As a function of Wavelength	
0/1	temperature		NX_FLOAT32		copy of temperature_env.sensor1.value
0/1		units	NX_CHAR	Kelvin	
0/1	temperature_log		NXlog		temperature_log.value is a link to temperature_env.sensor1.value_log.value
0/1	temperature_env		NXenvironment		Additional sample environment information
0/1	Magnetic_field		NXenvironment		copy of magnetic_field_env.sensor1.value
0/1		units	NX_CHAR	Tesla	
0/1	magnetic_field_env		NXenvironment		Additional sample environment information

Examples of other possible environment variables are: electric_field, conductivity, resistance, voltage, pressure, flow, stress, strain, shear, surface_pressure.

The temperature_log cannot be directly linked to temperature_env.sensor1.value_log as the names of the two entities are different; instead the contents of the two NXlog entries must be linked i.e. temperature_log.value to temperature_env.sensor1.value_log.value etc. For the same reason, “temperature” must be specified as a copy of “temperature_env.sensor1.value” rather than a link

The sample location is specified by the **position**, **shape** and **size** variables (see “size and shape” section in the introduction). The **size** variable gives the outer dimension of the sample in terms of its local (rotated) axes; if the sample is hollow, **inner_size** can be specified in the same way.
The **beam** member gives details of the incident beam direction (plus other optional information) on the sample. Combining this direction with the sample position allows the point of sample-beam interaction to be determined, which is important if the sample is larger than beam. While this method covers most simple shapes, a “general” shape would probably require specifying intersecting surfaces and an NXshape class.

The **path_length_window** variable is required for multiple scattering corrections when scattering from the can is taken into account. The straight through beam would traverse [path_length_window] + [path_length(sample)] + [path_length_window]. However, for complex samples **path_length** may have to be calculated from the sample shape for each detector.

Appropriate background correction for some sample environments is complicated by the fact that in the ABSENCE of the sample neutrons simply don’t get scattered through anything like the same range of angles as in the sample measurement. One way around this is to employ simulations, hence the inclusion of “simulated data” in the **Type** field.

Differences from Current NeXus Standard
The introduction of sample components, the NXenvironment class and size/shape variables for sample dimensions are new. Also symmetry_cell_settings has been renamed to unit_cell_class for consistency with the other unit_cell_* variable naming.

NXenvironment

Information about equipment used and the conditions that it imposes on e.g. the sample

RE	Name	Attribute	Type	Value	Description
	NXenvironment			Name of sample environment	
1	**type**		NX_CHAR	"cryostat" | "furnace" | "pressure cell" | "water bath" | "CCR"	Type of apparatus.
1	**name**		NX_CHAR	Apparatus identification code/model number	e.g. "OC100-011"
1	**short_name**		NX_CHAR	Alternative short name	SE name from ISIS scheduling database?
1	**description**		NX_CHAR	Description of the apparatus	e.g. "100mm bore orange cryostat with Roots pump"
1	**program**		NX_CHAR	Computer program controlling the apparatus	e.g. LabView VI name
0/1	position		NXposition	The position and orientation of the apparatus	
0/1		scanned	NX_BOOLEAN	0 | 1	
1	**notebook**		NXnotebook	Additional information	e.g. LabView logs, digital photographs, equipment setup details etc
1+	***{sensor1}***		NXsensor	First sensor	

Differences from Current NeXus Standard
This is a new class for storing "sample environment" information

NXsensor

RE	Name	Attribute	Type	Value	Description
	NXsensor			Name of sensor	
1	**model**		NX_CHAR	Sensor identification code/model number	
1	**name**		NX_CHAR	Name for the sensor	
1	**short_name**		NX_CHAR	Short name of sensor	e.g. “TEMP1”, the dashboard display
0/1	attached_to		NX_CHAR	“sample” | “can”	Where sensor is attached (“sample” if not present
0/1	position		NXposition		Defines the axes for logged vector quantities if they are not the global instrument axes
1	**measurement**		NX_CHAR	“temperature” | “pH” | “magnetic_field” | “electric field” | “conductivity” | “resistance” | “voltage” | “pressure” | “flow” | “stress” | “strain” | “shear” | “surface_pressure”	What the sensor measures, but can we get this uniquely from units instead?
1	**type**		NX_CHAR	*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”	Sensor hardware type
0/1	Run_control		NX_INT32[np]	-1=none, 0=value, 1=value_deriv1 etc	Indicates if data collection is synchronised with the sensor value
0/1		scanned	NX_BOOLEAN	0 | 1	
0/1	High_trip_value		NX_FLOAT32[np]	Upper control bound of sensor reading	Only if run control
0/1		Units	NX_CHAR		
0/1		scanned	NX_BOOLEAN	0 | 1	
0/1	Low_trip_value		NX_FLOAT32[np]	Lower control bound of sensor reading	Only if run control
0/1		Units	NX_CHAR		
0/1		scanned	NX_BOOLEAN	0 | 1	
1	**Value**		NX_FLOAT32[np,n]	nominal setpoint or average value	Setpoint or average value
1		**Units**	NX_CHAR		
0/1		scanned	NX_BOOLEAN	0 | 1	

0/1	Value_deriv1		NX_FLOAT32[np,n]	Nominal/average first derivative of value	We may run control on e.g. strain rate
0/1		Units	NX_CHAR		
0/1		scanned	NX_BOOLEAN	0 | 1	
0/1	Value_deriv2		NX_FLOAT32[np,n]	Nominal/average second derivative of value	
0/1		Units	NX_CHAR		
0/1		scanned	NX_BOOLEAN	0 | 1	
0/1	Value_log		NXlog	Time history of sensor readings	
0/1	value_deriv1_log		NXlog	Time history of sensor readings	
0/1	value_deriv2_log		NXlog	Time history of sensor readings	
0/1	External_field_brief		NXCHAR	“along beam”|”across beam”|”transverse”|”solenoidal” | ”flow-shear gradient”|”flow-vorticity”	“along beam” | ”across beam” | ”transverse” | ”solenoidal” | ”flow-shear gradient” | ”flow-vorticity”
0/1	External_field_full		NXorientation	For complex external fields not satisfied by External_field_brief	For complex external fields not satisfied by External_field_brief

Differences from Current NeXus Standard
This is a new class

Notes
Value is defined as NX_FLOAT32[np,n] - usually n=1 (scalar), but n=3(vector) and n=6 (symmetrical tensor) are also possible.
Value_log is a continuous time log over all {np} periods. While the setpoint/average value will be calculated (or recalculated if periods are cycled) separately for each period, to get the time dependence of a quantity the values of period_start_time and period_end_time from NXentry must be used and the correct time range extracted. If periods are cycled, this will not be possible as multiple period_start_times are not recorded.
There will be a NXenvironment group for every different piece of SE apparatus in use during the experiment for which the data is being collected; e.g. a pressure cell experiment might require the use of a *“vertical height stage”* (1 value to set/log), a *“horizontal translation stage”* (1 value to set/log), the *“pressure cell”* (1 pressure sensor to log), a *“temperature controller”* (2 temperature sensors to set/log), and possibly a *“fluid bath”* (1 temperature sensor to set/log) and/or a *“pH sensor”* (1 value to log), making a possible 6 instances of NXenvironment and 7 entries under NXlog

Some Recommended Units for Value
Temperature: “Kelvin” | “Celsius”
Ion-selective electrode: “ppm”|”Molar”
Magnetic field: “Tesla” (also “Gauss”|”Oested”?)
Electric field: “volts/metre”
Conductivity: “microsiemens/cm”|”micromho/cm”|”ppm”
Resistance: “ohms”|”kilohms”|”megaohms”
Voltage: “volts”|”millivolts”|”microvolts”|”kilovolts”
Pressure: “Pascals” (also “kilopascals”| “bar”| “kilobar”| “mm Hg”?)
Flow: “litres/min”|”ml/sec” (also “gallons/hour?)
Stress: “newtons/metre”|”pascals”
Strain: “percent”
Shear: “per second”
Surface pressure: “millinewtons/metre”

NXinstrument

Definition of instrument descriptions comprising various beamline components. Each component will be a NeXus group defined by its position relative to some origin (see "coordinates/positions" in the introduction).

RE	Name	Attribute	Type	Value	Description
	NXinstrument			Name of instrument	
1	**Source**		NXsource	Source	
0/1	Moderator		NXmoderator	Moderator	
0+	*{Aperture}*		NXaperture	Name of beamline aperture	
0+	*{Attenuator}*		NXattenuator	Name of beam attenuator	
0+	*{Chopper}*		NXchopper	Name of chopper	
0+	*{Collimator}*		NXcollimator	Name of collimator	
0+	*{Crystal}*		NXcrystal	Name of crystal analyser / monochromator	
0+	*{Flipper}*		NXflipper	Name of beam polarisation flipper	
0+	*{Guide}*		NXguide	Name of beam guide mirror	
0+	*{Polarizer}*		NXpolarizer	Name of beam polarizer	
1+	***{bank1}***		NXdetector	Name of detector bank	
0+	*{Monitor1}*		NXmonitor	Name of monitor	
0+	*{Beam_stop1}*		NXbeam_stop		
0/1	Type		NX_CHAR	"elastic","inelastic","direct etc.	
1	**Long_name**		NX_CHAR	Full name of instrument	e.g. "PRISMA"
1	**Short_name**		NX_CHAR	instrument abbreviated name	At ISIS, the 3 letter abbreviation e.g. "PRS"
0/1	Description		NX_CHAR	Description of instrument	
0/1	URL		NX_CHAR	Web address of description/manual	

Differences from Current NeXus Standard
NXmonitor has been moved to NXinstrument so there is no name clash with an NXdata used to store the monitor counts; also NXmirror has been renamed as the general NXguide. As well as some new classes (NXbeam_stop etc.) **long_name**, **short_name**, **description**, **type**, and **url** are introduced

Notes
The name of an NXdetector or NXmonitor instance in NXinstrument matches that of an NXdata instance in NXentry and corresponds to data from that detector/monitor.

NXsource

This class describes global properties of the beam as a whole as opposed to that part which reaches the sample (which is situated after "NXmoderator")

RE	Name	Attribute	Type	Value	Description
	NXsource			Name of source	
1	**Name**		NX_CHAR	Facility name	
1	**Type**		NX_CHAR	"Spallation Neutron Source" | "Pulsed Reactor Source" | "Reactor Neutron Source" | "Synchrotron X-ray Source"	
1	**Probe**		NX_CHAR	"neutrons" | "muons" | "x-rays"	
1	**Frequency**		NX_FLOAT32	Frequency of pulsed source at the target	"at target" allows for ISIS TS-2 where the main proton beam will be split with 1 in 5 pulses diverted to the second target
1		**Units**	NX_CHAR	Hertz	
1	**Power**		NX_FLOAT32	source power at target	
1		**Units**	NX_CHAR	Mega.watt	
1	**Current**		NX_FLOAT32	nominal source current at target	
1		**Units**	NX_CHAR	Micro.ampere	
1	**Voltage**		NX_FLOAT32	nominal source voltage at target	
1		**Units**	NX_CHAR	mega.electronvolt	
1	**target_material**		NX_CHAR	"W" | "Ta" | "Depleted U" | "Enriched U" | "Hg" | "Pb" | "C"	
0/1	Notes		NX_TEXT	Source/facility related messages or announcements during the experiment	At ISIS, the MCR beam messages
0/1	Pulse_width		NX_FLOAT32	Source pulse width	Proton pulse at ISIS
0/1		Units	NX_CHAR		
01/	pulse_shape		NXdata	Source pulse shape	

Differences from Current NeXus Standard
current and **voltage** are defined rather than **proton_current** and **proton_voltage** for generality; also **period** can be removed if **frequency** is taken to mean "frequency at target". The **probe** member is new and also is a **notes** member for storing any source/facility related messages. The old **moderator** member has been moved to a separate NXmoderator class

NXmoderator

Properties of the moderator that the instrument is looking at

RE	Name	Attribute	Type	Value	Description
	NXmoderator			Name of moderator	
1	**Position**		NXposition		
1	**Moderator**		NX_CHAR	"H20" | ”D20” | "H2" | "CH4"	
1	**Width**		NX_FLOAT32		
1		**Units**	NX_CHAR	cm	
1	**Height**		NX_FLOAT32		
1		**Units**	NX_CHAR	cm	
1	**Thickness**		NX_FLOAT32		
1		**Units**	NX_CHAR	cm	
1	**Angle**		NX_FLOAT32	angle of moderator face normal to incident beam	
1		**Units**	NX_CHAR	degree	
1	**poison_depth**		NX_FLOAT32		
1		**Units**	NX_CHAR	Cm	
1	**poison_material**		NX_CHAR	"Gd"|"Cd"|...	
0/1	temperature		NX_FLOAT32	temperature	
0/1		Units	NX_CHAR	Kelvin	
0/1	temperature_log		NXlog	log file of moderator temperature	
0/1	pulse_shape		NXdata	moderator pulse shape	

Differences from Current NeXus Standard
This is a new class – moderator information was formerly stored in NXsource

Notes
Store the resolution function as well?

NXaperture

Definition of a beamline aperture e.g. slits,jaws

RE	Name	Attribute	Type	Value	Description
	NXaperture			Name of aperture	Name of aperture
1	**position**		NXposition	location and orientation of aperture	
1		**Scanned**	NX_BOOLEAN	0 | 1	
1	**description**		NX_CHAR		
1	**shape**		NX_CHAR	"Rectangular" | "Circular" | "Elliptical" | “slit”	
1	**size**		NX_FLOAT32[np,6]	dimensions of aperture	
1		**units**	NX_CHAR		
0/1		scanned	NX_BOOLEAN	0 | 1	
0/1	beam		NXbeam	Details of incident beam	
0/1	material		NX_CHAR	Material used for aperture	

Differences from Current NeXus Standard
This class has been greatly simplified by use of **shape** and **size** variables as detailed in the introduction. For slits, the infinite dimension is indicated by a zero in the corresponding element of the size array

Notes
Think about motor positions....

NXdetector

Definition of a detector, detector bank, or multidetector

RE	Name	Attribute	Type	Value	Description
	NXdetector			Name of detector	
1	**description**		NX_CHAR	description/model	
1	**shape**		NX_CHAR	“cuboid” | “cylindrical”	Shape of each detector element; may need to be an array, so should change to NX_INT32
1	**spectrum_index**		NX_INT32[ns]	List of global spectrum numbers	Global spectrum numbers are unique across all NXdetector instances
1	**detector_index**		NX_INT32[ns]	DETECTOR_INDEX[i] is the location of first detector of spectrum SPECTRUM_INDEX[i] in the array DETECTOR_LIST[nd]	See below for full explanation of usage
0/1	detector_count		NX_INT32[ns]	DETECTOR_COUNT[i] is the total number of detectors forming spectrum SPECTRUM_INDEX[i]	If this is absent, assume 1 detector per spectrum
0/1	detector_list		NX_INT32[nd]	Sorted List of detector numbers for fast lookup	If this is absent, it is assumed to have elements rising sequentially from 1 to nd
0/1	detector_code		NX_INT32[nd]	A unique user supplied code number for each detector	Used to indicate e.g. the bank or location. May want this [nda] instead
0/1	detector_wiring		NX_INT32[nlines,7]		Alternative to using crate/slot/input
0/1	Crate		NX_INT32[nd]	The crate number	detector card number at ISIS
0/1	Slot		NX_INT32[nd]	slot number	At ISIS the module (DIM) number
0/1	Input		NX_INT32[nd]	input number	At ISIS the module position input
1	**time_of_flight**		NX_FLOAT32[ntc+1]	time channel bin boundaries	linked to NXdata

		units		micro.second	
1	**primary_flight_path**		NX_FLOAT32	distance from “timing point 0” to scattering point	The ISIS L1 value
1		**Units**	NX_CHAR	Metre	
1		**Calibration_status**	NX_CHAR	“nominal” | “measured”	
0/1	Distance		NX_FLOAT32[np,nda]	Flight path length from scattering point to detector	The secondary flight path (L2) value at ISIS
1		**Scanned**	NX_BOOLEAN	0 | 1	
1		**Units**	NX_CHAR	Metre	
1		**Calibration_status**	NX_CHAR	“nominal” | “measured”	
1	**array_type**		NX_INT32	0 | 1 | 2	For 0 (non array), 1D or 2D array. This determines whether angles are calculated or supplied.
0/1	two_theta		NX_FLOAT32[np,nda]	Scattering (Bragg) angle of detector element	
1		**Scanned**	NX_BOOLEAN	0 | 1	
1		**Units**	NX_CHAR	Degree	
1		**Calibration_status**	NX_CHAR	“nominal” | “measured”	
0/1	azimuthal_angle		NX_FLOAT32[np,nda]		
1		**Scanned**	NX_BOOLEAN	0 | 1	
1		**Units**	NX_CHAR	Degree	
1		**Calibration_status**	NX_CHAR	“nominal” | “measured”	
0/1	solid_angle		NX_FLOAT32[np,nda]	Solid angle subtended by detector at sample	
1		**Scanned**	NX_BOOLEAN	0 | 1	
1		**Units**	NX_CHAR	steradians	
1		**Calibration_status**	NX_CHAR	“nominal” | “measured”	
1	**Type**		NX_CHAR[nda]	He3 gas cylinder | He3 PSD" | "He3 multidetector" | "BF3 gas" | "scintillator" | "fission chamber" | “ZnS scintillator PSD”	
1	**size**		NX_FLOAT32[nda,3]	Size of detector element	
1		**units**	NX_CHAR		
0/1	gas_pressure		NX_FLOAT32[nda]	Detector gas pressure	
1		**Units**	NX_CHAR	Bar	
1	**absorption_cross_section**		NX_FLOAT32[nda]		
1	**Dead_time**		NX_FLOAT32[nda]		Delay before detector can count again
		units	NX_CHAR	micro.second	

1	**Hold_off**		NX_FLOAT32[nda]		Delay in detector registering event
		units	NX_CHAR	micro.second	
1	**Efficiency**		NXdata	Efficiency v Wavelength	
0/1	beam		NXbeam	Information on incident beam direction	
1	**pixel_type**		NX_INT32[nda]		
0/1	Pixels_x		NX_INT32	number of pixels along local x coordinate	
0/1	Pixels_y		NX_INT32	number of pixels along local y coordinate	
0/1	Pixel_size_x		NX_FLOAT32[pixels_x+1]		Use array for rise-time coded detectors
0/1		units		mm	
0/1	Pixel_size_y		NX_FLOAT32[pixels_y+1]		
0/1		units		mm	
0/1	Resolution				
0/1	position		NX_FLOAT32[np,nda,6]	position and orientation of the centre of (reference) detector element	Contents as for NXposition

Arrays of size [nd] or [nda] e.g. gas_pressure can be given as size [1], in which case they apply to all elements. If the origin of global coordinates is taken at the "scattering centre", then specifying **position** in spherical polar coordinates gives the equivalent information to (distance, two_theta, azimuthal _angle). For use of the nda array variable see the section on array detectors below; for the moment assume nda = nd.

The **detector_code** variable is a way to assign a reference number to a detector for diagnosis purposes; these number should be unique, but need not be contiguous. For example you may number all bank 1 detectors 1XXXX etc

At ISIS, data is collected per spectrum rather than per detector; often there will be a one to one mapping between the two, but detectors can be ganged together and so (ns <= nd). The proposed detector <-> spectrum indexing scheme is as follows. There are nd detectors {i} numbered i=[1,nd]. These detectors will have been attached to crate[i], slot[i]and input[i] of the electronics and have scattering angle two_theta[i] etc The output from these nd detectors will be mapped into ns spectra. As the global "spectrum number" must unique amongst all monitors and detectors the spectrum_index[j] array gives the ns unique global spectrum numbers for this NXdetector instance. To map between global spectrum number and detector we use the detector_list, detector_index and detector_count arrays. The detector_list array contains a list of detector numbers {i}, but they are arranged such that detectors which map to the same spectrum number appear sequentially. Spectrum spectrum_index[j] will thus have detector_count[j] detectors mapped into it, the actual detector numbers being given by detector_list[k],detector_list[k+1] … detector_list[k+detector_count[j]-1] where k = detector_index[j]

For comparison, if we used the same indexing method as the existing ISIS RAW file, we would not need detector_list, detector_index or detector_count; instead we would have spectrum_index[nd] which would directly give the spectrum number for a given detector. Though the old RAW file scheme is simpler, there is no direct information about how many and which detectors map to a given spectrum – you need to search the spectrum_index[nd] array each time to determine this.

While we will generally have only one NXdetector instance, one case in which we would need two is if multiple time regimes (not all detectors with the same time_of_flight array) was implemented. In ISIS

DAE2 each detector card in the acquisition electronics can, in fact, have a different set of time channel boundaries.

The NXdata corresponding to the detector bank has the same name as the detector bank itself (no name clash as they are at different levels in the NeXus hierarchy). An alternative would be to have a link to the NXdata directly in the NXdetector?

Two detector mapping schemes are provided: (crate,slot,input) or (detector_wiring). If the detector_wiring variable is present, it will contain nlines of the following 7 integers which can be used to generate (crate,slot,input) arrays:

Crate	Slot	input_start	input_increment	detector_start	detector_increment	detector_end

For example the two lines:

1	1	1	1	1	1	16000
1	2	1	1	16001	1	32000

would assign the inputs 1 to 16000 on (crate=1,slot=1) sequentially to detector numbers 1 to 16000 and the same inputs on (crate=1,slot=2) to detector numbers 16001 to 32000. For large detector arrays, this mapping scheme presents a considerable space saving.

Array Detectors (array_type variable > 0)
For a bank of one dimensional position sensitive array detectors, or a full two dimensional detector, two_theta etc. can be calculated from the detector geometry and need not be stored in the file. The presence of a PSD is indicated by the **array_type** variable and, though there are still nd detectors in the bank, the **nda** variable is no longer equal to **nd**

Value of array_type variable	**Value of nda**	**Meaning**
0	nd	two_theta etc. supplied for all nd detectors
1	>=1	We have nda linear PSD detectors and will supply one value for each tube; other values will be calculated. The local x axis is defined to be along a tube and pixels_y=1
2	1	We have a 2D PSD and will calculate all scattering parameters.

The reference point (origin of axes) used by the **position** variable for a PSD detector is taken to be the centre of the bottom right detector element/pixel as viewed from the moderator. Detector numbers raster along the x axis (i.e. by row if there is no rotation) from bottom right to top left as viewed from the moderator; we have chosen right to left so increasing detector number follows increasing x by our axes convention. To instead raster by column or from left to right you merely need to specify a rotation in the **position** variable. From the position of the reference element and the pixel size, it is possible to calculate all scattering parameters for the array.

Differences from Current NeXus Standard
polar_angle has been removed as it was part of the spherical polar coordinate description of the location of a detector element – this is now covered by **position**. Instead we now include **two_theta** and have generalised **distance** to "secondary flight path length" for the detector.

Notes
The **position** variable of a detector element is only sufficient to calculate scattering parameters if the neutron follows a direct path to it from the scattering point. In other cases **distance** and **two_theta** will need to be supplied – often these will have been obtained via calibration ("measured") rather than calculation.

Need to find place for efficiency_file, coordinate_file_x, coordinate_file_y, mask_file, resolution_file..

Need to add detector_code into detector_wiring scheme

NXmonitor

Definition of monitor data; it is similar to the NXdetector group but also include integrals, or scalar monitor counts, which are often used in both in both pulsed and steady-state instrumentation.

In addition to the values listed here, any variables defined in NXdetector can also be specified.

RE	Name	Attribute	Type	Value	Description
	NXmonitor				
1	**detector**		NXdetector	Any members of NXdetector may be used	
0/1	Integral		NX_FLOAT32	Integral of monitor spectrum	
0/1	Range		NX_FLOAT32[2]	Range integral performed over	
0/1		units	NX_CHAR	micro.second	
0/1	Integral_log		NXlog	Log of monitor integral	As per ISIS Beam log process
0/1	area_sampled		NX_FLOAT32	Proportion of beam sampled	Do we want a % or an absolute area?

Differences from Current NeXus Standard
area_sampled is new

Notes
To interpret beam efficiency it is necessary to know what proportion of the beam is sampled by the monitor, hence the “area_sampled” variable. An alternative would be to include an NXbeam member and then work this out from the position of the monitor.
The data for the monitor is stored in an NXdata member with the same name as the monitor.

NXchopper

Definition of a beamline chopper, e.g., disk chopper or Fermi chopper.

RE	Name	Attribute	Type	Value	Description
	NXchopper				
1	**Position**		NXposition	Description of chopper	
1	**Type**		NX_FLOAT32[:]	"Fermi" |"disk" | "counter rotating disk" | "double disk"	
0/1	hole_shape		NX_CHAR	"rectangle"	
0/1	hole_size		NX_FLOAT32[3]		
1	**Frequency**		NX_FLOAT32[:]		positive frequency gives anti-clockwise rotation about z
1		**Units**	NX_CHAR	Hz	
1	**frequency_log**		NXlog		
1	**Radius**		NX_FLOAT32	Radius of chopper	
1		**Units**	NX_CHAR	Cm	
0/1	Curvature		NX_FLOAT32	Radius of curvature of fermi chopper	
0/1	Slit_width		NX_FLOAT32	Width of fermi chopper slits	
0/1		Units	NX_CHAR	Cm	
0/1	Blade_width		NX_FLOAT32	Width of fermi chopper blades	
0/1		Units	NX_CHAR	Cm	
0/1	Slit_number		NX_INT32	Number of fermi chopper slits	
0/1	Energy		NX_FLOAT32	Energy transmitted by chopper	
0/1		Calibration_status	NX_CHAR	nominal | measured	
0/1	Delay				
0/1	Trigger_log		NXlog	Log of trigger pulses	
0/1	phase		NX_FLOAT32	Nominal/specified Chopper phase	
0/1	Phase_log		NXlog	Log of chopper phases	
0/1	Tilt_angle				
0/1	Sync_signal		NX_CHAR	Chopper synchronisation source	e.g. SMP
0/1	Opening_angle				For double disk choppers

0/1	absorbing_material		NX_CHAR		for fermi chopper
0/1	transmitting_material		NX_CHAR		for fermi chopper

Differences from Current NeXus Standard
phasing_log has been renamed to phase_log to tie up with “phase”. Period has been removed as it is directly related to frequency. New items include: sync_signal, opening angle

Notes
A counter rotating chopper is described by two instances of NXchopper; **frequency** has been defined to include a sense of rotation, so + and – values will be used in the two instances

NXdae

Special details of the data acquisition electronics used. The class will be highly institute specific and was created as a place to keep information about the running data acquisition system in case the NeXus file was used as a parameter file by the computer control program. It could also be used to store information useful for diagnostic purposes.

RE	Name	Attribute	Type	Value	Description
	NXdae				
1	**Clock_frequency**		NX_CHAR		
1	**Frames_per_period**		NX_INT32	Frames for each hardware period	
1	**Period_map**		NX_INT32[:]	Hardware period map array	
1	**Vetos**		NXveto		Define NXveto class?
1	**Frame_sync_source**		NX_CHAR	TOF | Internal | External	TOF | Internal | External
1	**trigger_source**		NX_CHAR	internal | external	
1	**Dae_memory**		NX_INT32		
1	**type**		NX_CHAR	"DAE1","DAE2"	"DAE1","DAE2"
1	**interface**		NX_CHAR	"SCSI" | "VME"	
1	**Veto_Frames**		NX_INT32[]		
1	**notes**		NX_TEXT	Any special notes	
1	**poslut**		NX_INT32[]	position lookup table	DAE2 detector position lookup table
1	**run_status**		NX_INT32	Current run state	0=setup, 1=running, -1=paused, -2=waiting
1	**monitor_spectrum**		NX_INT32	spectrum number to display on dashboard	
1	**current_period**		NX_INT32		
1	**period_type**		NX_INT32		0=software, 1=hardware
1	**frame_sync_delay**		NX_FLOAT32	frame sync delay	in clock cycles

Differences from Current NeXus Standard
New class

Notes
We may store ISIS spectrum 0 here. Also check on TCM and TCP.

NXcollimator

Definition of a beamline collimator.

RE	Name	Attribute	Type	Value	Description
	NXcollimator				
1	**Position**		NXposition	Location and orientation of centre of collimator	
1	**Type**		NX_CHAR	soller | radial | oscillating | honeycomb	
1	**Length**		NX_FLOAT32	(vane) Length of collimator	
1		**Units**		Cm	
1	**Soller_angle**		NX_FLOAT32	Angular deflection of soller collimator	Angular deflection of soller collimator
1		**Units**		milli.radians	
1	**Horizontal_aperture**		NX_FLOAT32	Front Horizontal aperture (if rectangular)	
1		**Units**		cm	
1	**Vertical_aperture**		NX_FLOAT32	Front Vertical aperture (if rectangular)	
1		**Units**		cm	
1	**Radius**		NX_FLOAT32	Front Radius of aperture (if circular)	
1		**Units**		milliradians	milliradians
1	**Divergence_x**				
1	**Divergence_y**				
0/1	frequency		NX_FLOAT32	Frequency of oscillating collimator	
0/1	frequency_log		NXlog		
0/1	blade_thickness		NX_FLOAT32		
0/1	blade_spacing		NX_FLOAT32		
0/1	absorbing_material		NX_CHAR		
0/1	transmission_material		NX_CHAR		
0/1	entrance_shape		NX_CHAR		
0/1	exit_shape		NX_CHAR		

Differences from Current NeXus Standard

Notes
For radial collimators the "aperture" size at the front combined with vane length and the angular divergence in both planes you can calculate the exit aperture.
Should use shape/size convention instead of “radius” and “horizontal_aperture”

NXattenuator

Definition of a beamline attenuator.

RE	Name	Attribute	Type	Value	Description
	NXattenuator				
1	**Position**		NXposition		
1	**description**		NX_CHAR	Description of attenuator	
1	**Thickness**		NX_FLOAT32	Along beam direction	Along beam direction
1		**Units**	NX_CHAR	Cm	
1	**Scattering_cross_section**		NX_FLOAT32	Coherent + incoherent	
1		**Units**	NX_CHAR	Barns	
1	**Absorption_cross_section**		NX_FLOAT32		
1		**Units**	NX_CHAR	Barns	
1	**Transmission**		NXdata		
1	**Width**		NX_FLOAT32		
1		**Units**	NX_CHAR	cm	
1	**height**		NX_FLOAT32		
1		**Units**	NX_CHAR	cm	
1	**Radius**		NX_FLOAT32		
1		**Units**	NX_CHAR	cm	
1	**material**		NX_CHAR	“Pb” | “Polythene” | “Perspex”	

Differences from Current NeXus Standard

width, height, radius new

Notes
redefine to use “shape” and “size[]” variables instead?

NXbeam

Definition 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.

RE	Name	Attribute	Type	Value	Description
	NXbeam				
0/1	incident		NXposition	Beam direction on entering beamline component	beam along (0,0,+z) in local axes
0/1	final		NXposition	Beam direction on leaving beamline component	
0/1	incident_shape		NX_CHAR	“elliptical” | “rectangular”	Shape of incident beam cross-section
0/1	incident_size		NX_FLOAT32[2]	dimensions of incident beam	meaning depends on shape
0/1	Incident_energy		NX_FLOAT32[:]	Energy on entering beamline component	
0/1	Final_energy		NX_FLOAT32[:]	Energy on leaving beamline component	
0/1	Energy_transfer		NX_FLOAT32[:]	Energy change caused by component	
0/1	Incident_wavelength		NX_FLOAT32[:]		
0/1	Final_wavelength		NX_FLOAT32[:]		
0/1	Incident_polarisation		NX_FLOAT32[i,3]		
0/1	Final_polarisation		NX_FLOAT32[i,3]		
0/1	Flux		NX_FLOAT32[i]	Flux incident on beam plane area	Flux incident on beam plane area
0/1	Spectrum		NXdata	Distribution of beam with respect to relevant variable e.g. wavelength	Distribution of beam with respect to relevant variable e.g. wavelength
0/1	divergence_x		NX_FLOAT32		
0/1	divergence_y		NX_FLOAT32		

The path of the beam is described by an NXposition object for consistency with other components. The NXdistance member of NXposition gives a reference point through which the beam passes; the beam travel down its local z axes, which is rotated from the global coordinate system in a way specified by the NXorientation member.

Differences from Current NeXus Standard
incident, final, incident_shape, incident_size are new; however do we need incident_* and final_*? Should an NXbeam not just describe the state of the beam at a given position?

Notes
We may need to allow the SCANNED attribute as this could be referred to from movable components

NXbeam_stop

RE	Name	Attribute	Type	Value	Description
	NXbeam_stop				
1	**description**		NX_CHAR		
1	**Position**		NXposition	Distance and orientation	
1	**Shape**		NX_CHAR	circular | square	
1	**Type**				
1	**Width**				
1	**Height**				
1	**Diameter**				
1	**thickness**				
1	**material**				
1	**In_use**		NX_BOOLEAN		

Differences from Current NeXus Standard
new class

Notes
Should recode to use “shape” and “size” variables

NXcrystal

Crystal monochromator or analyser

RE	Name	Attribute	Type	Value	Description
	NXcrystal				
1	**Position**		Nxposition	Location of crystal	Location of crystal
1	**Wavelength**		NX_FLOAT32	Optimum diffracted wavelength	Optimum diffracted wavelength
1		**Units**	NX_CHAR	Angstrom	
1	**Energy**		NX_FLOAT32	Optimum diffracted energy	Optimum diffracted energy
1		**Units**	NX_CHAR	MeV	
1	**Lattice_parameter**		NX_FLOAT32	Lattice parameter of the nominal reflection	Lattice parameter of the nominal reflection
1		**Units**	NX_CHAR	Angstrom	
0/1	lattice_parameter_error		NX_FLOAT32		
1	**Reflection**		NX_INT32[3]	[hkl] for nominal reflection	[hkl] for nominal reflection
1	**Horizontal_curvature**		NX_FLOAT32	Horizontal curvature of focusing crystal	Horizontal curvature of focusing crystal
1		**Units**	NX_CHAR	Degree	
1	**Vertical_curvature**		NX_FLOAT32	Vertical curvature of focusing crystal	Vertical curvature of focusing crystal
1		**Units**	NX_CHAR	Degree	

1	**Horizontal_aperture**		NX_FLOAT32	Horizontal aperture, if rectangular	Horizontal aperture, if rectangular
1		**Units**		Cm	
1	**Vertical_aperture**		NX_FLOAT32	Vertical aperture, if rectangular	Vertical aperture, if rectangular
1		**Units**	NX_CHAR	cm	
0/1	mosaic_spread		NX_FLOAT32[3]		
0/1	temperature_log		NXlog		
0/1	shape		NX_CHAR		
0/1	size		NX_FLOAT32[3]		
0/1	description		NX_CHAR		
0/1	cut_angle		NX_FLOAT32		

Differences from Current NeXus Standard

Notes
need expert input

Should also allow for periods and SCANNED

NXguide

Definition of a beamline guide -

RE	Name	Attribute	Type	Value	Description
	NXguide				
1	**position**		NXposition		
1	**length**		NX_FLOAT32		
1	**entrance_size**		NX_FLOAT32[2]		x,y
1	**exit_size**		NX_FLOAT32[2]		x,y
1	**type**		NX_CHAR		
1	**details**				
1	**mode**				
1	**incident_angle**				
1	**reflectivity**		NXdata	Reflectivity as function of wavelength [nsurf,i]	
1	**horizontal_bend_angle**		NX_FLOAT32		
1	**vertical_bend_angle**		NX_FLOAT32		
1	**interior_atmosphere**		NX_CHAR	“vacuum”	inside guide
1	**external_material**		NX_CHAR		outside substrate
1	**m_value**		NX_FLOAT32[nsurf]		
1	**substrate_material**		NX_FLOAT32[nsurf]		
1	**substrate_thickness**		NX_FLOAT32[nsurf]		
1	**coating_material**		NX_FLOAT32[nsurf]		
1	**substrate_roughness**		NX_FLOAT32[nsurf]		
1	**coating_roughness**		NX_FLOAT32[nsurf]		
1	**coating_material**		NX_FLOAT32[nsurf]		
1	**number_sections**		NX_INT32	number of substrate sections	

What should be the convention of which order the surfaces are stored ?
[top, bottom, left, right] ?

Do we need to include the entrance/exit windows ?

Differences from Current NeXus Standard
This is a new more general class, encompassing the old NXmirror

Notes
needs expert input

NXpolarizer

Definition of a beamline spin polarizer

RE	Name	Attribute	Type	Value	Description
	NXpolarizer				
1	**Position**		NXposition		
1	**Type**			mirror | He3	
1	**Details**				
1	**Mode**				
1	**Incident_energy**				
1	**M_value**				
1	**Reflectivity**		NX_FLOAT32[i]	Reflectivity as function of wavelength	
1	**Efficiency**		NX_FLOAT32[i]	Efficiency as function of wavelength	
1	**Polarisation**		NX_FLOAT32[i]	Polarisation as function of wavelength	
1	**Relaxation_time**				
1	**Path_length**				

Differences from Current NeXus Standard
all new

Notes
need expert input – can we merge with NXflipper?

NXflipper

Definition of a beamline spin flipper

RE	Name	Attribute	Type	Value	Description
	NXflipper				
1	**type**				
1	**Position**		NXposition		
1	**Length**		NX_FLOAT32		
1	**Efficiency**		NX_FLOAT32[i]	Efficiency as function of wavelength	
1	**State**		NXlog		

Differences from Current NeXus Standard
all new

Notes
need expert input – can we merge with NXpolarizer?

NXdistance

A class to specify the location of a component in either the global coordinate system ("absolute"), or in the coordinate system of another component ("relative") For absolute positioning we use the MCSTAS convention of:

- Z axis points down the beam
- X axis is perpendicular to the beam in the horizontal plane, pointing left as seen from the source
- Y axis points upwards perpendicular to the beam in the vertical plane

The origin of absolute coordinates is taken at the scattering centre, which will be at (or near) the sample position.

RE	Name	Attribute	Type	Value	Description
	NXdistance				
0/1	Absolute		NX_BOOLEAN	Absolute=true, relative=false	If relative, need to follow link in the distance member
1	**Type**		NX_CHAR	cartesian | cylindrical | spherical	
0/1	Distance		NXdistance	Link to other object if we are "relative", else absent	
1	**Value**		NX_FLOAT32[np,3]	(X,y,z), (r,theta,z) or (r,theta,phi).	We must use metres and degrees as there are no units attributes due to possible mixture of data types
0/1		scanned	NX_BOOLEAN	0 | 1	

Differences from Current NeXus Standard
New class

NXorientation

A class to specify the orientation of the local component's (x,y,z) axes either absolutely (using the global coordinate frame) or relative to another component's axes. There are many choices for the three (euler) angles required to specify this [14] as well as other schemes such as direction cosines.

RE	Name	Attribute	Type	Value	Description
	NXorientation				
0/1	Absolute		NX_BOOLEAN	Absolute=1, relative=0	If relative, need to follow link in the orientation member
1	**Type**		NX_CHAR	"euler_zyx"	Convention for angles / information
0/1	Orientation		NXorientation	Link to another object if we are relative, else absent	
1	**Value**		NX_FLOAT32[np,3]	The orientation information	would need to be [np,9] if we allow direction cosines
1		**Units**	NX_CHAR	Degree | dimensionless	dimensionless if "cosine"
0/1		Scanned	NX_BOOLEAN	0 | 1	

Differences from Current NeXus Standard
New class

Notes

NXposition

A class to specify the position (location + orientation) of a component either absolutely or relative to another component. We need to split "position" into "distance" and "orientation" classes to allow linking of the members (linked members must have the same name)

RE	Name	Attribute	Type	Value	Description
	NXposition				
0/1		scanned	NX_BOOLEAN	0 | 1	We vary position with data collection period number if this is TRUE
0/1		is_origin	NX_BOOLEAN	0 | 1	Indicates that we are an origin rather than attached to an instrument component
1	**distance**		NXdistance		
0/1	orientation		NXorientation		No rotation if absent
0/1	description		NX_CHAR		Description of position; normally only used if this is an origin

Differences from Current NeXus Standard
New class

NXnote

This is a convenience class for storing additional information that might be attached to another class in a NeXus file. To store a collection of notes, see NXnotebook. The original idea was borrowed from Cooper et al [4] – our extension is to add the "Mime content-type" entry, so a program can then use any built-in file associations on the reading computer to invoke the correct external display program for an image, video, audio etc.

RE	Name	Attribute	Type	Description	Description
	NXnote				
0/1	Author		NX_CHAR	Author of note	
0/1	Date		ISO8601	Date note created/added	
1	**Type**		NX_CHAR	Mime content-type of note data field	e.g. text/plain, image/jpeg etc
0/1	File_name		NX_CHAR	Name of original file name	Present if note was read from an external source
0/1	Description		NX_CHAR	Title of an image or other details of the note	
1	**Data**		NX_BINARY	Binary note data.	If this is text, the line terminator should be \r\n as in NX_TEXT

Differences from Current NeXus Standard
New class

Notes

NXnotebook

A class for storing a collection of notes; purely text notes could be stored as NX_TEXT instead

RE	Name	Attribute	Type	Value	Description
	NXnotebook				
1		count	NX_INT32	Number of notes	
1+	***{Note1}***		NXnote	note data	

Differences from Current NeXus Standard
New class

Notes
Labelling notes note1, note2 etc eases in later access and sequencing

Acknowledgements

I would like to thank all members of the ISIS NeXus Working group [2] for their help and input during the drafting of this document. In particular, Toby Perring for doing the initial groundwork, Steve King for helping formulate the NXsample class and Richard Heenan for suggestions on handling array detectors.

References

1. The NeXus web site, http://www.neutron.anl.gov/NeXus/
2. The ISIS NeXus Working Group (INWG) – F. A. Akeroyd, M. J. Bull, S. I. Campbell, S. P. Cottrell, M. R. Daymond, D. W. Flannery, M. Gutmann, W. S. Howells, S. M. King, K. J. Knowles, C. M. Moreton-Smith and T. G. Perring (isis_nexus@isise.rl.ac.uk)
3. The NeXus definitions and discussion SWIKI, http://www.neutron.anl.gov:8080/NeXus/
4. "NeXus file specification issues" Gary Cooper, Tom Kozlowski, Thomas Proffen (Los Alamos Neutron Scattering Centre, May 2002) http://strider.lansce.lanl.gov/canps/nexus/nexus_spec_xml_1.pdf
5. "The sasCIF dictionary", http://www.embl-hamburg.de/ExternalInfo/Research/Sax/sascif.html
6. The IUCR CIF data format (http://www.iucr.org/cif/)
7. "The Application of the NeXus data format to ISIS Muon Data", D. Flannery, S. P. Cottrell and P. J. C. King (RAL-TR-2001-029)
8. NeXus standards committee http://www.neutron.anl.gov/nexus/NeXus_advisers.html
9. The NeXus XML Meta-DTD Format for specifying class definitions, http://www.neutron.anl.gov/NeXus/NeXus_metaformat.html
10. The HDF file format from NCSA, http://hdf.ncsa.uiuc.edu/
11. The Unidata UDUNITS package, http://www.unidata.ucar.edu/packages/udunits/ (the units definitions file is at http://www.unidata.ucar.edu/packages/udunits/udunits.dat)
12. ISO 8661 time specification format http://www.cl.cam.ac.uk/~mgk25/iso-time.html and http://www.w3.org/TR/NOTE-datetime
13. W R Busing & H A Levy, Acta Cryst, (1967), 22, 457-464.
14. H. Goldstein, "Classical Mechanics" (Addison-Wesley)