The NeXus API and Utilities

Freddie Akeroyd
STFC ISIS Facility

Introduction

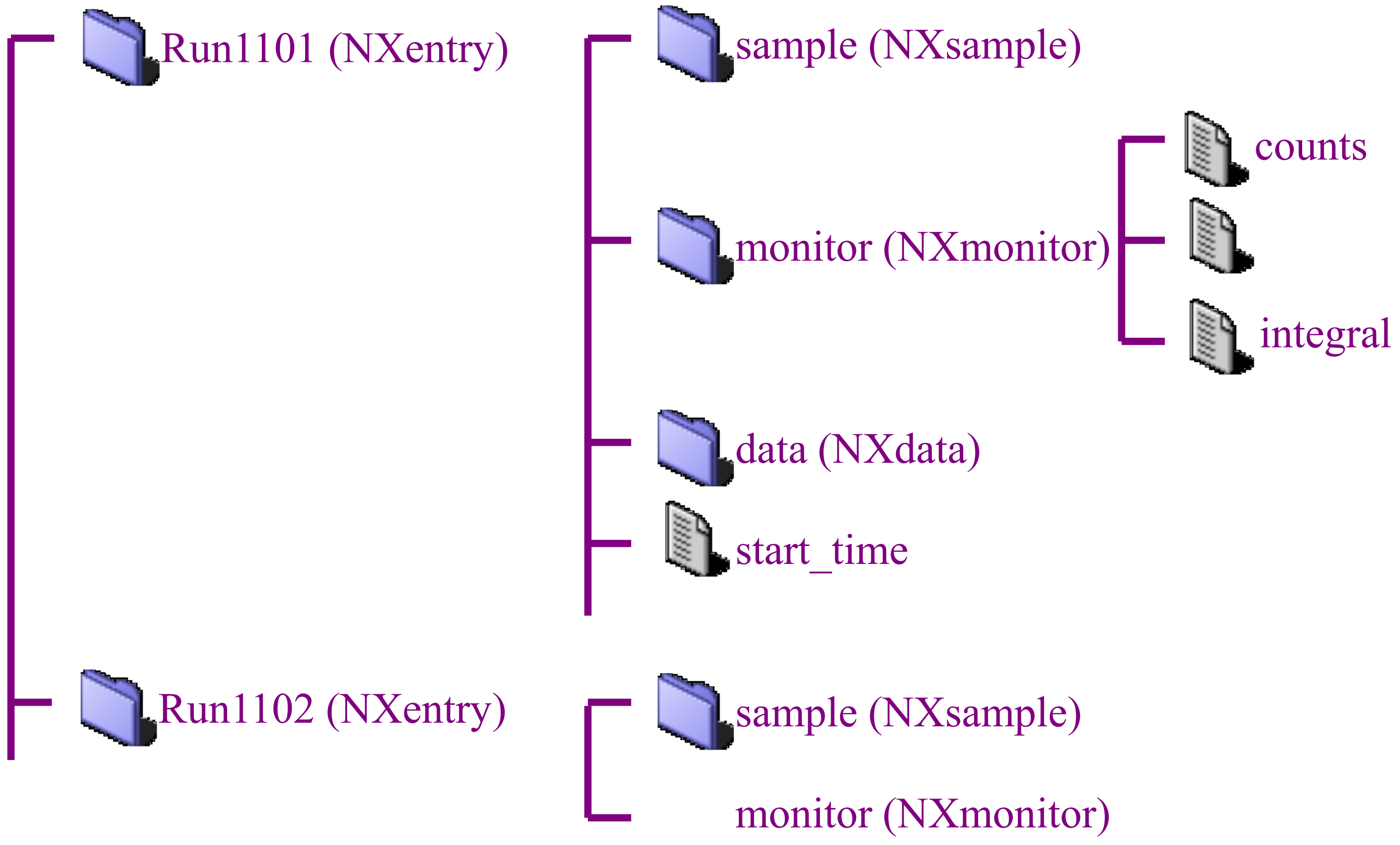
- Mark has covered NeXus rules
    - ➢How objects are arranged in a hierarchy
    - ➢Rules for storing objects
    - ➢Metadata e.g. units, axes
- We will now cover
    - ➢How objects are written and stored in files
    - ➢Utilities for manipulating NeXus files

NeXus Files

- NeXus hierarchy and objects are stored in files
- However data sets vary in size (10kb – 1GB+)
- So the same hierarchy can be represented differently in the low level (byte) file content
  - Currently as ASCII XML or binary HDF4/HDF5
  - Allows user choice of storage efficiency (binary) v text editor readability (ASCII)

Hierarchical Structure of a NeXus file

It looks similar to a file system



The NeXus API (NAPI)

- Designed to make it easier to read and write NeXus files
- Hides unnecessary features / complexity of underlying file system storage commands
- A uniform interface to hide knowledge of the underlying low level storage format
- Enforces certain elements of the standard
- Core API in C, bindings in other languages

NeXus Objects

There are only three types of NeXus object

- Data
  - scalar or multidimensional arrays
  - integer (1, 2, 4, or 8 bytes), real (single or double), or character
- Groups
  - folders containing sets of data items and/or other groups
  - A group is an instance of a “NeXus classes”
  - Have both a name and a class (type)
  - All Official NeXus classes have the NX prefix
  - Items within a group are referred to as “entries” by the API
- Attributes
  - meta-data attached to a data item *e.g.* Units, axes labels

What the API does underneath

- Maps objects into their low-level representation equivalents
  - e.g. In HDF4 NeXus Data -> SDS (Scientific Data Set), NeXus Group -> Vgroup
- Adds additional bookkeeping information
  - NeXus class for HDF5 (in XML/HDF4 it is part of the representation)
  - Additional linking metadata (to make following links easier)

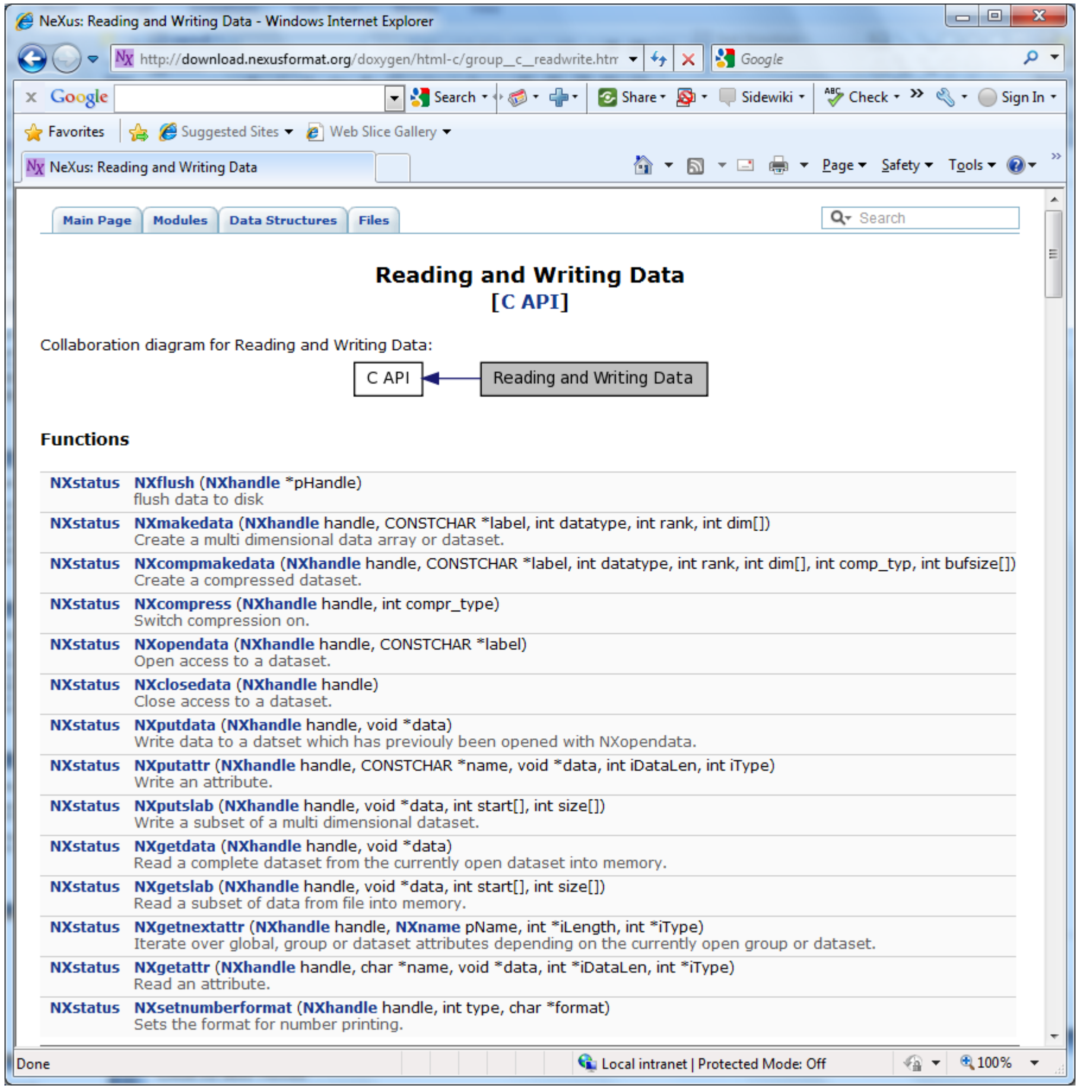
API Overview

- Written in C, but with additional language bindings (C++,JAVA,PYTHON,IDL,...)
- open, close, get, put style interface
  - like walking a file system tree
- Includes “get next” attribute / entry calls
- Also enquire array dimensions

NeXus C API

- Doxygen documented at http://download.nexusformat.org/doxygen/html-c/
- All functions defined in file "napi.h"
- Also contains functions, typedefs and defines

NeXus API Documentation


NeXus: Reading and Writing Data - Windows Internet Explorer
http://download.nexusformat.org/doxygen/html-c/group__c__readwrite.htm
Main Page
Modules
Data Structures
Files
Search
Done
Local intranet | Protected Mode: Off
100%

Reading and Writing Data
[C API]

Collaboration diagram for Reading and Writing Data:

$$\begin{tikzcd}\text{C API} & \text{Reading and Writing Data} \arrow[l]\end{tikzcd}$$

Functions

NXstatus	NXflush (NXhandle *pHandle) flush data to disk
NXstatus	NXmakedata (NXhandle handle, CONSTCHAR *label, int datatype, int rank, int dim[]) Create a multi dimensional data array or dataset.
NXstatus	NXcompmakedata (NXhandle handle, CONSTCHAR *label, int datatype, int rank, int dim[], int comp_typ, int bufsize[]) Create a compressed dataset.
NXstatus	NXcompress (NXhandle handle, int compr_type) Switch compression on.
NXstatus	NXopendata (NXhandle handle, CONSTCHAR *label) Open access to a dataset.
NXstatus	NXclosedata (NXhandle handle) Close access to a dataset.
NXstatus	NXputdata (NXhandle handle, void *data) Write data to a datset which has previouly been opened with NXopendata.
NXstatus	NXputattr (NXhandle handle, CONSTCHAR *name, void *data, int iDataLen, int iType) Write an attribute.
NXstatus	NXputslab (NXhandle handle, void *data, int start[], int size[]) Write a subset of a multi dimensional dataset.
NXstatus	NXgetdata (NXhandle handle, void *data) Read a complete dataset from the currently open dataset into memory.
NXstatus	NXgetslab (NXhandle handle, void *data, int start[], int size[]) Read a subset of data from file into memory.
NXstatus	NXgetnextattr (NXhandle handle, NXname pName, int *iLength, int *iType) Iterate over global, group or dataset attributes depending on the currently open group or dataset.
NXstatus	NXgetattr (NXhandle handle, char *name, void *data, int *iDataLen, int *iType) Read an attribute.
NXstatus	NXsetnumberformat (NXhandle handle, int type, char *format) Sets the format for number printing.

NeXus C typedefs

- NXhandle
  - Opaque structure identifying a particular file
  - Created by NXopen, passed to other routines, deleted by NXclose
- NXaccess
  - Enumeration of file access modes for NXopen
  - read only, read/write, create hdf5, create xml, ...
- NXstatus (NX_OK, NX_ERROR, NX_EOD)
- NXlink (Information about data/group links)

NeXus Defines

- Data type specifiers
  - NX_INT32, NX_FLOAT32, ...
- Other parameters
  - NX_MAXRANK – maximum array rank
  - NX_UNLIMITED – appendable array dimension

API Function Groups

- General Initialisation and shutdown
- Reading and writing groups
- Reading and writing data
- Meta data routines
- Linking
- Memory allocation
- External linking

Open and close file

NXhandle fileid;

If (NXopen(“file.nxs”, NXACC_CREATE5, &fileid) != NX_OK) { error }

If (NXclose(&fileid) != NX_OK) { error }

If opening existing file, uses NX_LOAD_PATH environment variable.

“fileid” is passed to all other API functions

Create a group

```
NXhandle fileid;  /* NXopen previously called */
/* error checking for NX_OK skipped */
NXmakegroup(fileid, “entry1”, “NXentry”);
NXopengroup(fileid, “entry1”, “NXentry”);
/* do something with group */
NXclosegroup(fileid);
```

Create a data item

```
NXhandle fileid;  /* NXopen already called */
/* group created or opened from before */
int rank= 2; int dims[] = { 100, 100 };
int counts_array[100][100];
NXmakedata(fileid, “counts”, NX_INT32, rank, dims);
NXopendata(fileid, “counts”);
NXputdata(fileid, counts_array);
NXputattr(fileid, “axes”, “[px,py]”, 7, NX_CHAR);
NXclosedata(fileid);
```

Read a data item

```
NXhandle fileid;  /* NXopen already called */
/* NXopendata already called */
int datatype, rank, dims[NX_MAXDIMS], *counts;
NXgetinfo(fileid, &rank, dims, &datatype);
NXmalloc(&counts, rank, dims, datatype);
NXgetdata(fileid, counts);
/* do something with counts */
NXfree(&counts);
```

Data slabs

Can read or write a portion of a larger dataset using NXgetslab and NXputslab

```
NXhandle fileid;  /* NXopen already called */
/* counts dataset opened from before */
int counts_slice[10][10];
int start[] = { 5, 5}, size[] = { 10, 10 } ;
NXgetslab(fileid, counts_slice, start, size);
```

Appending to arrays

- Can define a array that “grows”
- Specify NX_UNLIMITED as the slowest varying dimension
- Call NXputslab to write each section

Compressing data

- Use NXcompmakedata rather than NXmakedata
- Specify a compression type (e.g. NX_COMP_LZW)
- Specify the dimensions of a "compression chunk"
  - A chunk must be read or written in one go by the underlying software
  - Trade off final data size V read/write speed
- Use NXsetcache to improve HDF5 performance
  - At the cost of more program memory
- Read compressed data the same way as uncompressed data

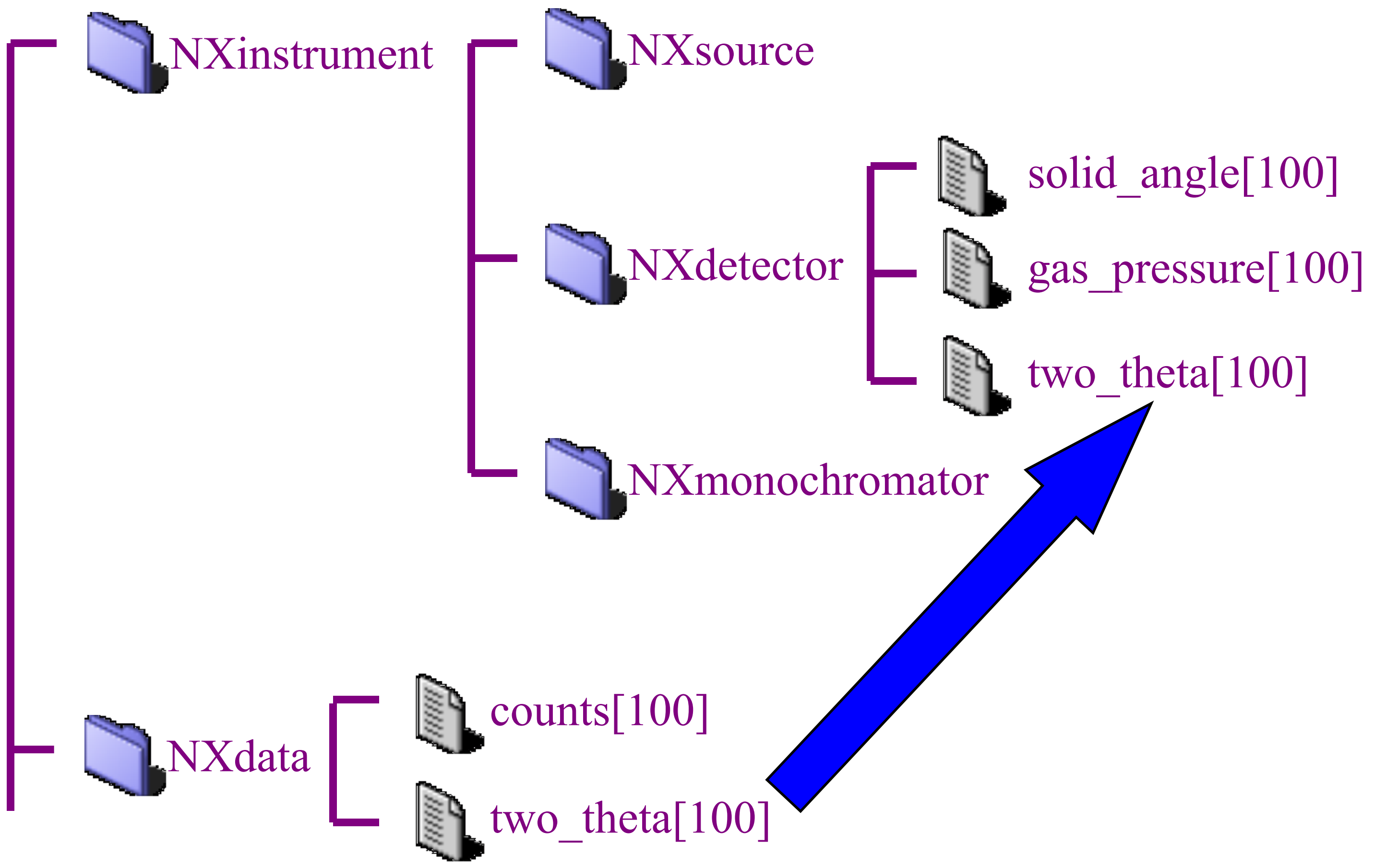
Faster navigation

- Go straight to an object and open
  - NXopenpath(fileid, “/path/to/item”)
- Open the group containing an object
  - NXopengrouppath(fileid, “/path/to/item”)
- Open source group of linked dataset
  - NXopensourcegroup (fileid)

Linking items

- Items in a file can be linked
- They appear at multiple place is the hierarchy, but take no additional space
  - Like symbolic links in a filesystem
- Target item may have a different name
- Use NXopensourcegroup to open the parent group of the link target
  - E.g. If you linked two_theta from NXdetector into NXdata, this would open NXdetector group for you

Data Linking



Link example

```
NXlink dlink;
/* open dataset */
NXgetdataID(fileid, &dlink);
/* navigate to another location */
NXmakenamedlink(fileid, “newname”, &dlink);
```

Similarly use NXgetgroupID for linking groups

External Linking

- Can create a group linked to an external URL
  - Only a local file is currently supported
  - nxfile://filename/path/to/group
  - Also uses NX_LOAD_PATH to find “filename”
- Use NXlinkexternal() to create group
- Use NXisexternalgroup() if you wish to test if a group is external
- Nxopengroup() works as normal

Iterating through groups

- Call NXinitgroupdir to reset search list
- Call NXgetnextentry to return name, class and datatype of each item
  - returns NX_EOD when no more items
- If it is a group, “name” and “class” are set
- If it is a data rather than group item, “class” is “SDS” and datatype is set

Iterating attributes

- Call NXinitattrdir to reset search list
- Call NXgetnextattr to return name, size and datatype of each item
  - returns NX_EOD when no more items
- Call NXgetattr to read attribute contents

High Level API

- NXU routines
  - Utility routines for finding axes and combining make/open/close functionality
  - Originally developed in F90 (todo: port to C) yet
- NXdict
  - Dictionary access API
  - Define file structure and item alias in configuration file
  - Write data items via dictionary API calls

Core C++ API

- Thin layer on top of C API
- Throws exceptions for NX_ERROR conditions
- Uses std::vector<type> rather than arrays
- Uses std::string rather than char*
-

C++ Example

```
std::vector<int> counts;
try{
    NeXus::File nf(“test.nxs”,NXACC_CREATE);
    nf.makeGroup(“entry1”, “NXentry”, true);
    nf.writeData(“counts”, counts);
}
```

C++ Streams API

- IOStream like C++ interface
- “nf << item” to write; “nf >> item” to read
- Best shown by example

C++ Streams API Write Example

```
// create entry and data item in new file
    std::vector<double> w;
   NeXus::File nf(“test.nxs”,NXACC_CREATE5);
   nf << Group(“entry1”, “Nxentry”) << Data(“dat1”,w,
   ”int_attr”, 3);
   nf.close();
// add additional item
   NeXus::File nf1(“test.nxs”,NXACC_RDWR);
   nf1 >> Group(“entry1”, “Nxentry”) >> Data(“dat1”) <<
   Attr(“double_attr”, 6.0)
   nf1.close();
```

C++ Streams API Read Example

```
double d; int i; std::vector<double> w1;
NeXus::File nf(“test.nxs”,NXACC_CREATE5);
nf >> Group(“entry1”, “NXentry”) >>
    Data(“dat1”, w1, “int_attr”, i, “double_attr”, d);
```