The NeXus API and Utilities

Freddie Akeroyd
STFC ISIS Facility
for NeXus Workshop, PSI
10 – 12 May 2010

Introduction

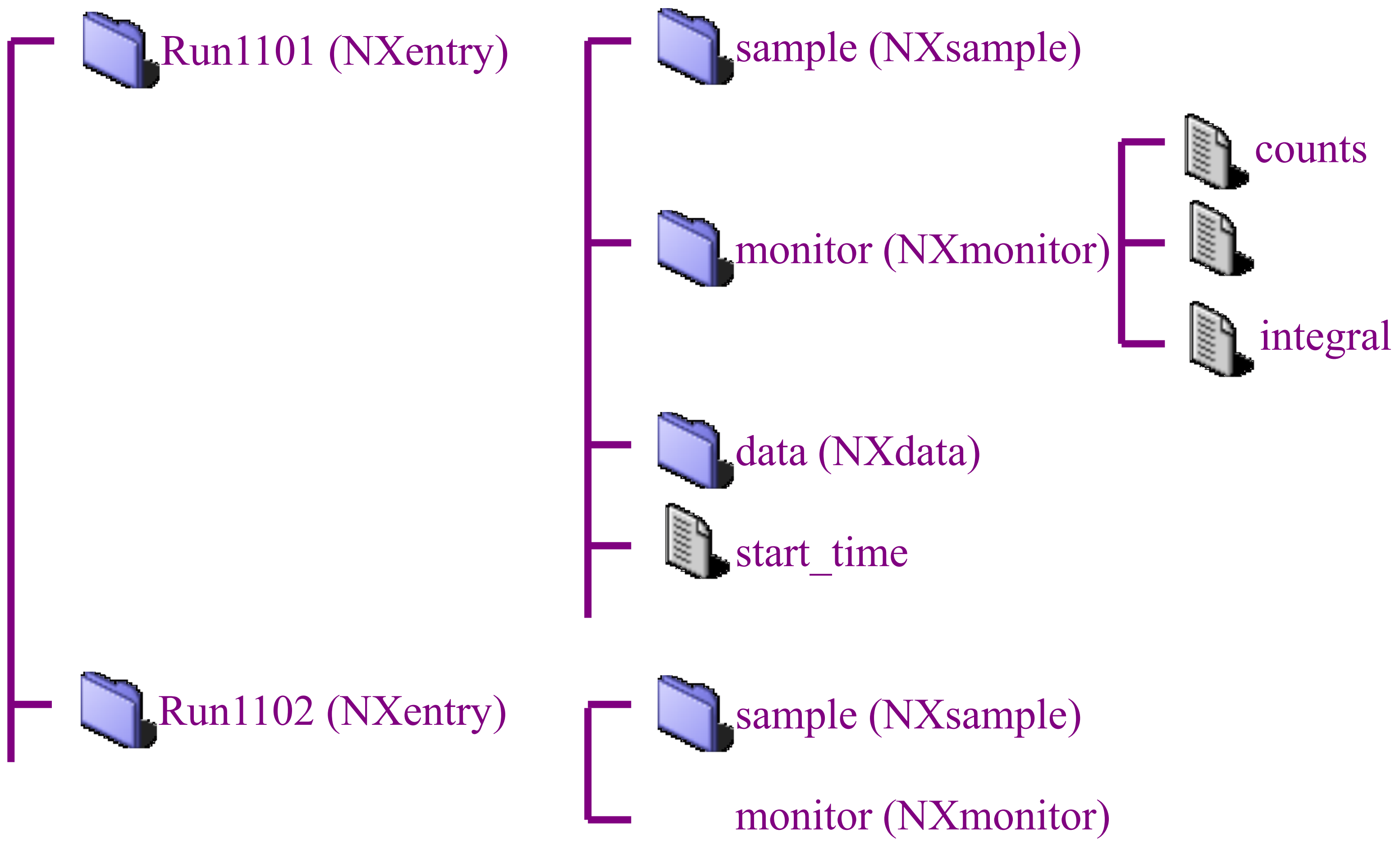
- Mark has covered NeXus rules
  - How objects are arranged in a hierarchy
  - Rules for storing objects
  - Metadata e.g. units, axes
- We will now cover
  - How objects are written and stored in files
  - Utilities for manipulating NeXus files

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
  - e.g. NeXus class identifier for HDF5 (in XML/HDF4 it is part of the representation)
  - Additional linking metadata (to make tracing links easier)

API Overview

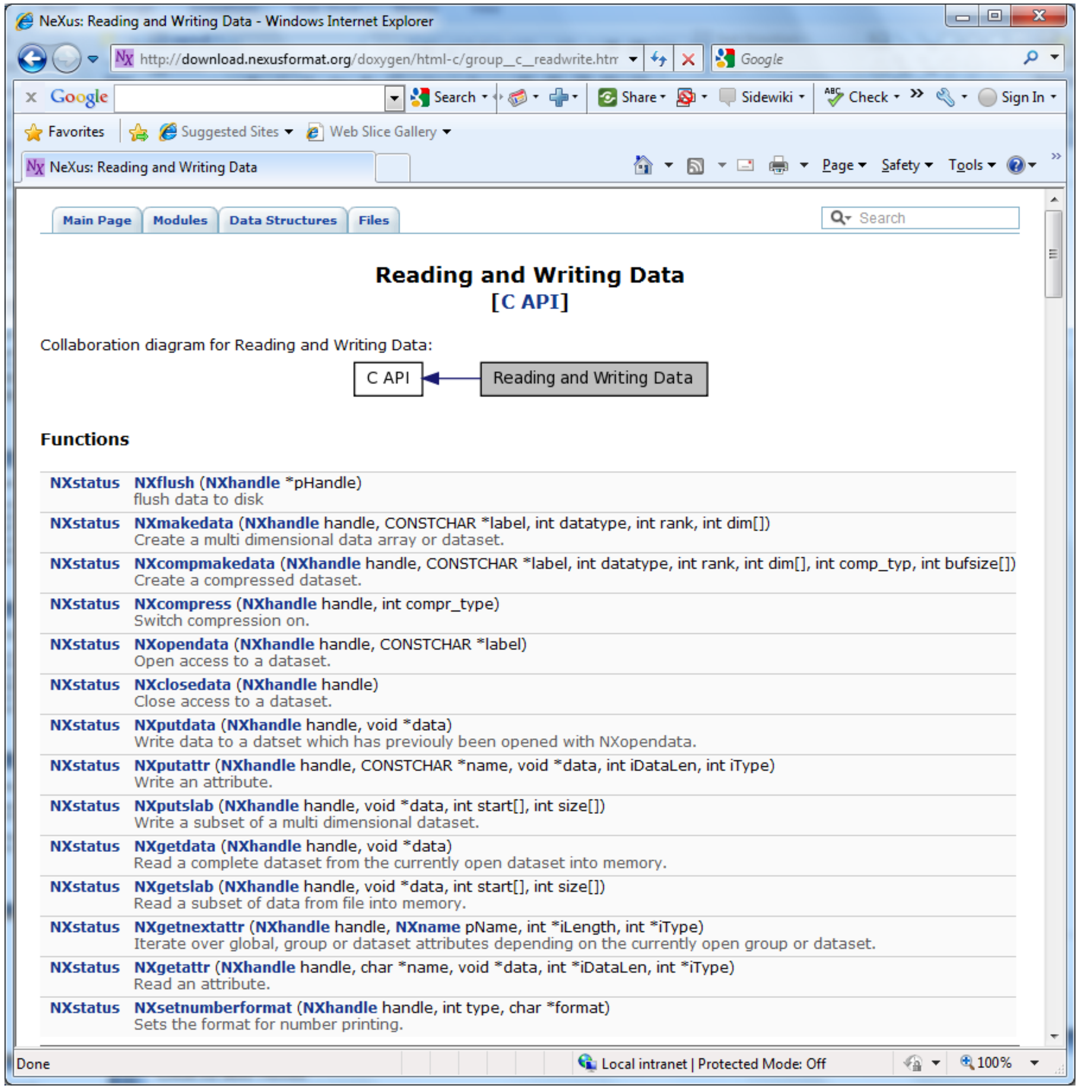
- Written in C, but with additional language bindings (C++,JAVA,PYTHON,IDL,...)
- open, close, get, put style interface
  - like walking a file system tree
- Includes “get next” attribute / entry calls
- Also enquire array dimensions

NeXus C API

- This API and bindings are documented at
  http://download.nexusformat.org/
- All functions defined in file “napi.h”
- Also contains typedefs, constants, defines etc.

C API Doxygen Documentation

http://download.nexusformat.org/doxygen/html-c/


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:

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

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 useful constants/parameters
  - NX_MAXRANK – maximum array rank
  - NX_UNLIMITED – appendable array dimension

C 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 (C API)

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

NeXus C++ API

- As most people here have O-O experience, the C++ API will be used for remaining examples
  - NXhandle encapsulated in NeXus::File class
  - C API functionality exposed as member functions
- A few difference / extensions from C API
  - throws exceptions on NX_ERROR conditions
  - supports std::vector<type> and std::string
  - can obtain a container of entry/attribute details rather than needing to iterate via “get next”

Open and Close file (C++ API)

```
// can use nf.open() instead of constructor
try{
        NeXus::File nf(“test.nxs”,NXACC_CREATE5);
        …
        // file closed by destructor or nf.close()
}
```

If opening an existing file, uses NX_LOAD_PATH environment variable for path searching.

C++ Example

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

Create a group (C++ API)

```
NeXus::File nf(“test.nxs”,NXACC_CREATE5);

nf.makeGroup(“entry1”, ”NXentry”); // just create
nf.openGroup(“entry1”, ”NXentry”);  // move inside

// To create and enter a group in one call use
nf.makeGroup(“sample”, ”NXsample”, true);

// now leave the groups
nf.closeGroup();     // leave sample
nf.closeGroup();     // leave entry1
```

Create a data item (C++ API)

```
NeXus::File nf(“test.nxs”, NXACC_CREATE5);
std::vector<int> dims;
dims.push_back(100); dims.push_back(100);
int counts_array[100][100];
nf.makeData( “counts”, NX_INT32, dims, true);
nf.putData(counts_array);
nf.putAttr( “axes”, “[px,py]”);  // attribute of “counts”
nf.closeData();                   // leave “counts”
std::vector<int> px;              //  fill px with numbers
nf.writeData(“px”, px);           // make,open,put,close
```

Read a data item (C++ API)

```
NeXus::File nf(“test.nxs”, NXACC_READ);
// call openGroup() and openData()
std::vector<int> px;
nf.getData(px);
// alternative approach
int* counts;
NeXus::Info info = nf.getInfo();  // type and dims
nf.malloc(counts, info);
nf.getData(counts);
nf.free(counts);
```

Data slabs (C++ API)

Can read or write a portion of a larger dataset using getSlab() and putSlab()

```
std::vector<int> start, size;
// set start to {5, 5}  and size to {10, 10 }
int counts_slice[10][10];
nf.getSlab(counts_slice, start, size);
```

Appending to arrays

- Can define an array that “grows”
- Specify NX_UNLIMITED as the slowest varying dimension
- Call putSlab() to write each section

Compressing data

- Use compMakeData rather than makeData
- 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 setCache to improve HDF5 performance
  - At the cost of more program memory
- Reading compressed data is identical syntax to reading uncompressed data

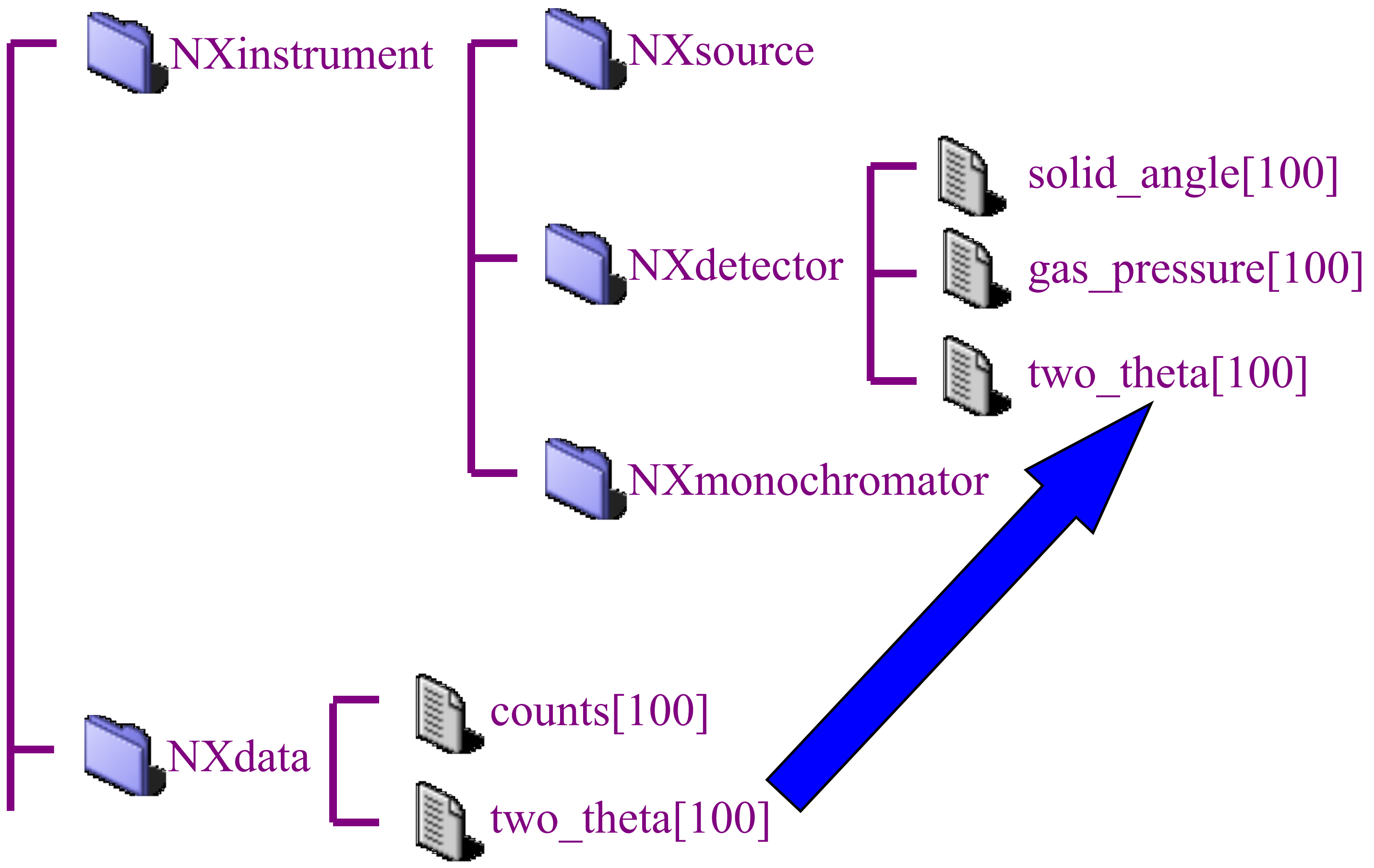
Enhanced File Navigation

- Go straight to an object and open
  - openPath(“/path/to/item”)
- Open the group containing an object
  - openGroupPath(“/path/to/item”)
- Open source group of linked dataset
  - openSourceGroup()

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 openSourceGroup 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

```
NeXus::File nf
NXlink dlink;
// open dataset
nf. getDataID(&dlink);
// navigate to another location
nf. makeNamedLink(“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 linkExternal() to create group
- Use isExternalGroup() if you wish to test if a group is external
- openGroup() works as normal

Iterating Through Groups (C++ API)

```
using namespace std;
NeXus::File nf;
typedef map<string, string> smap_t;
 smap_t   entries;
// navigate to somewhere
entries = nf.getEntries();
for(smat_t::const_iterator it=entries.begin(); it !=
entries.end(); ++it) {
   cout << “Name “ << it->first <<
              “ class “ << it->second << endl;
}
```

Iterating through groups (C API)

- 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 (C++ API)

```
using namespace std;
NeXus::File nf;
vector<NeXus::AttrInfo> attrinfo;
// navigate to an item
attrinfo = getAttrInfos ();
for(int i=0; i<attrinfo.size(); ++i) {
	NeXus::AttrInfo& info = attrinfo[i];
	cout << “Name “ << info.name << endl;
	// info.type is Nxnumtype enumeration
	// use nf.getAttr() to read value
}
```

Iterating attributes (C API)

- 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

C++ Streams API

- Alternative IOStream like C++ interface
- “nf << item” to write; “nf >> item” to read/navigate
- 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_READ);
nf >> Group(“entry1”, “NXentry”)
    >> Data(“dat1”, w1, “int_attr”, i, “double_attr”, d);
```

JAVA API

- Uses files jnexus.jar and libjnexus.so (jnexus.dll on windows)
- The *org.nexusformat* namespace defines a NeXusFile object with usual API methods
- Throws NeXusException on error
- When writing strings (NX_CHAR) need to pass the bytes via String.getBytes()
- Call NeXusFile.finalise() to explicitly close file

JAVA API (cont)

- JAVA API has additional functions:
  - groupdir() returns contents of group as HashTable
    (key = name, value = class)
  - attrdir() returns attribute details as HashTable
    (key = name, value = AttributeEntry object containing type and length)

JAVA Example

```
import org.nexusfromat.*;
int iData1[][] = new int[3][10];
int iDim[] = new int[2];
NeXusFile nf = new NeXusFile(“test.txs”,
      NeXusFile.NXACC_CREATE5);
nf.makegroup(“entry1”,”Nxentry”);
nf.opengroup(“entry1”,”Nxentry”);
iDim[0] = 3; iDim[1] = 10;
nf.makedata(“data1”,NeXusFile.NX_INT32,2,iDim);
nf.opendata(“data1”);
nf.putdata(iData1);
```

Python API

- Thin binding to NeXus C API
- nxs module open() returns NeXus file object
- Can use strings rather than constants
  - ‘r’ -> nxs.ACC_READ, ‘float32’ -> nxs.NX_FLOAT32
- getdata() / putdata () use numpy array objects
- NeXus class has C API functions plus extras:
  - entries() returns dictionary of group contents
  - attrs() returns dictionary of attributes

Python Example (write)

```
import nxs,numpy
nxfile = nxs.open(‘test.nxs’,’w5’)  #  nxs.ACC_CREATE5
nxfile.makegroup(“entry”,”NXentry”)
nxfile.opengroup(“entry”,”NXentry”)
# val is a numpy double array
nxfile.makedata(“r8_data”,val.dtype,val.shape)
nxfile.opendata(“r8_data”)
nxfile.putdata(val)
nxfile.closedata()
```

Python Example (read)

```
import nxs,numpy
nxfile = nxs.open(‘test.nxs’,’r’)
nxfile.openpath(‘/entry/r8_data’)
val = nxfile.getdata()
nxfile.close()
```

For online documentation type:

```
help(nxs.napi)
```

Or visit http://download.nexusformat.org/

Building a C/C++ Program (linux)

Can be as simple as

```
gcc –o test test.c –lNeXus
```

But probably need to add appropriate –I and –L

Alternatively use the supplied nexus-config command (similar to pkg-config)

```
gcc  `nexus-config –libs –cflags`  -o  test  test.c
```

Building a C/C++ Program (windows)

- Install NeXus via windows installer kit
  - Usually to C:\Progam files\NeXus Data Format
  - Kit includes HDF DLLs etc.
  - Creates a NEXUSDIR environment variable
  - Also updates LIB, PATH and INCLUDE
- Link against the libNeXus.dll.lib import library in $(NEXUSDIR)\lib
- libNeXus-0.dll used at runtime

NeXus tools

- nxbrowse - CLI NeXus browser
- nxsummary - create summary of nexus file
- nxdiff - compare two files
- nxconvert – change the low-level representation of a NeXus file (e.g. HDF5 -> XML, HDF4 -> HDF5)
- nxtranslate - assembles NeXus file from other files (NeXus importer)
- nxextract - convert from NeXus to ASCII and binary (NeXus exporter)
- nxvalidate - validates file against definition
- nxplot – generic plotter for NeXus files

NXbrowse

- Simple command line browser with readline support
  - CD, DIR, and DUMP commands
  - Pressing <TAB> after typing a command or partial item name lists/completes that name
  - Try in “hands on” session

NXsummary

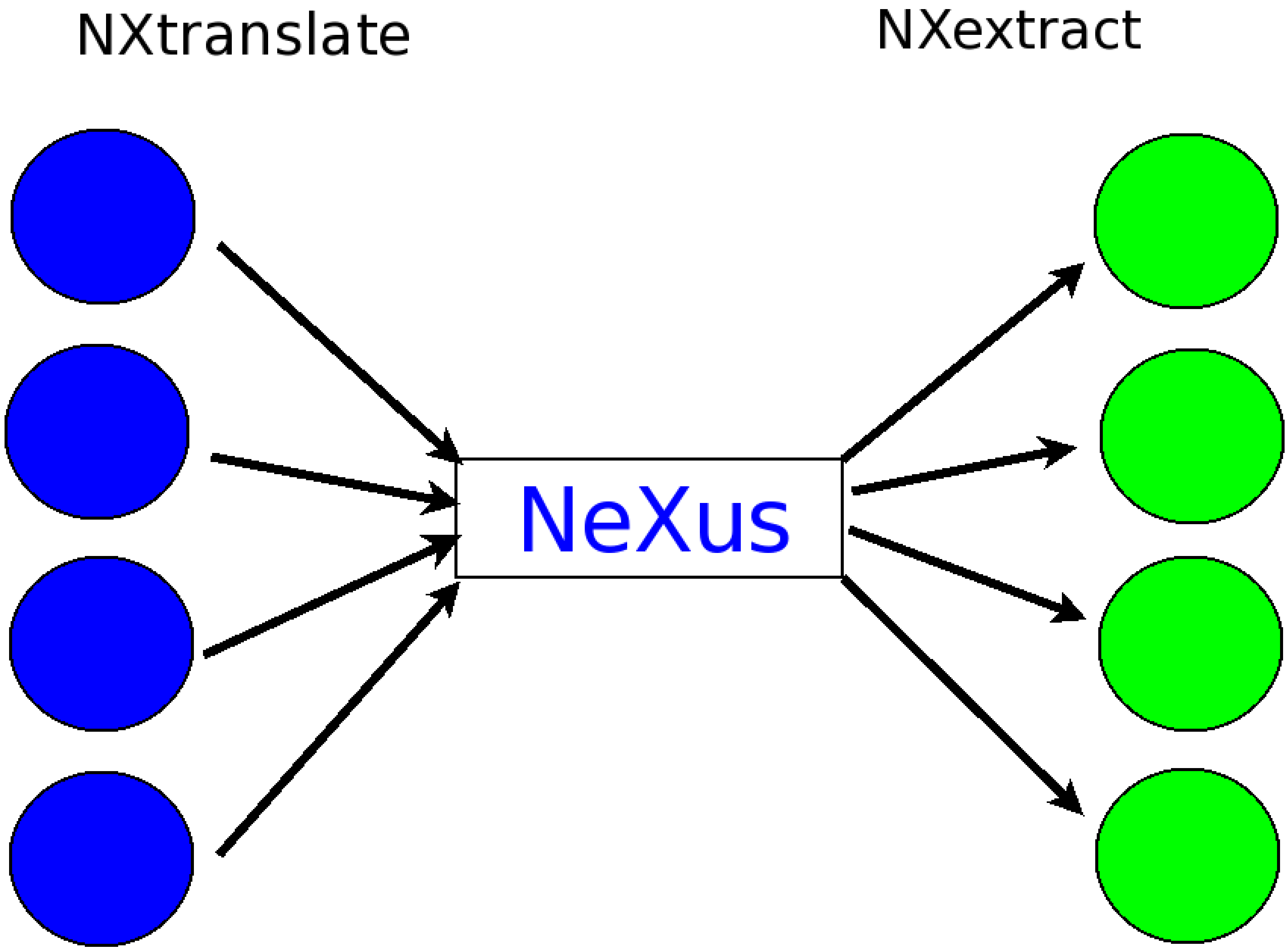
- Creates a quick summary of a NeXus file
- Items to be printed specified in a configuration file
- Optional XML output
- Supports simple transforms of items
  - e.g. count, sum

NXdiff

- Python program to compares two NeXus files
  - Example of using the NeXus python API
  - Type `nxdiff –help` for options

NXconvert

- Changes the low level representation of a NeXus file
  - e.g. HDF5 -> XML, HDF4 -> HDF5
  - Type nxconvert –help for options


NXtranslate
NXextract
NeXus

NXtranslate

- Anything to NeXus converter (importer)
- Placeholder for items specified in XML file
- Calls plugins to read and insert the appropriate data files into the output
  - Can handle FRM2, IPNS, SPEC, ESRF EDF, XML, binary floats/ints, ASCII column text and other NeXus files
  - Easily extended by writing your own plugin

Format of a translation file

- The file is similar to a NeXus XML data file, but with some additional attributes to insert data from plugins
  – NXS:mime_type specifies the plugin to use
  – NXS:source specifes a string to pass to the plugin to initialise it
  – NXS:location specifies a string to pass to the plugin to return data
- Manual at http://download.nexusformat.org/

Example translation file

```
<!– simple example to insert a title from one NeXus file
    into another →
<NXroot>
    <entry1 type="NXentry" NXS:source="other_file.nxs"
    NXS:mime_type="application/x-NeXus">
      <value1 my_attr="test">this is a test</ value1>
      <value2 NXS:location="/entry3/title" />
     </entry1>
</Nxroot>
```

NXextract

- Exports portions of a NeXus file as binary or ASCII
- Process controlled by configuration file
- Built-in syntax for looping through elements
- Contributed by Stephane Poirier, SOLEIL

Hands on Session

- Various examples installed on Linux server
- See details on the following document
- <to finish>

Additional /Old Slides follow

Create a group (C API)

```
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 (C API)

```
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 (C API)

```
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);
```

Example NeXus program in C

```
#include "napi.h"
int main()
{
  int counts[1000][50], n_t, n_p, dims[2], i;
  float t[1000], phi[50];
  NXhandle file_id;
/* Read in data using local routines */
  getdata (n_t, t, n_p, phi, counts);
/* Open output file and output global attributes */
  NXopen ("OUTPUT.HDF", NXACC_CREATE, &file_id);
  NXputattr (file_id, "user_name", "Joe Bloggs", 10, NX_CHAR);
  NXopengroup (file_id, "Entry1", "NXentry");
  NXopengroup (file_id, "Data1", "NXdata");
/* Output time channels */
  NXputdata (file_id, "time_of_flight", NX_FLOAT32, 1, n_t, t, “microseconds”);
/* Output detector angles */
  NXputdata (file_id, "phi", NX_FLOAT32, 1, n_p, phi, “degrees”);
/* Output data */
  dims[0] = n_t;
  dims[1] = n_p;
  NXputdata (file_id, "counts", NX_INT32, 1, dims, counts, “counts”);
  NXclosegroup (file_id);
  NXclose (file_id);
  return;
}
```