

NeXusDataFormat

1

Generated by Doxygen 1.7.4

Fri Feb 10 2012 18:38:27

Contents

1	NeXus API documentation	1
1.1	Purpose of API	1
1.2	Core API	1
2	Module Index	3
2.1	Modules	3
3	Module Documentation	5
3.1	C API	5
3.2	Data Types	5
3.3	General Initialisation and shutdown	5
3.3.1	Function Documentation	6
3.3.1.1	NXclose	6
3.3.1.2	NXopen	6
3.3.1.3	NXreopen	7
3.4	Reading and Writing Groups	7
3.4.1	Function Documentation	7
3.4.1.1	NXclosegroup	7
3.4.1.2	NXmakegroup	7
3.4.1.3	NXopengroup	8
3.5	Reading and Writing Data	8
3.5.1	Function Documentation	9
3.5.1.1	NXclosedata	9
3.5.1.2	NXcompmergedata	9
3.5.1.3	NXcompress	10
3.5.1.4	NXflush	10

3.5.1.5	NXgetattr	11
3.5.1.6	NXgetdata	11
3.5.1.7	NXgetnextattr	11
3.5.1.8	NXgetslab	12
3.5.1.9	NXmakedata	12
3.5.1.10	NXopendata	12
3.5.1.11	NXputattr	13
3.5.1.12	NXputdata	13
3.5.1.13	NXputslab	13
3.5.1.14	NXsetnumberformat	14
3.6	General File navigation	14
3.6.1	Function Documentation	15
3.6.1.1	NXgetnextentry	15
3.6.1.2	NXgetpath	15
3.6.1.3	NXinitattrdir	15
3.6.1.4	NXinitgroupdir	16
3.6.1.5	NXopengrouppath	16
3.6.1.6	NXopenpath	16
3.6.1.7	NXopensourcegroup	16
3.7	Meta data routines	17
3.7.1	Function Documentation	17
3.7.1.1	NXgetattrinfo	17
3.7.1.2	NXgetgroupinfo	18
3.7.1.3	NXgetinfo	18
3.7.1.4	NXgetrawinfo	18
3.7.1.5	NXgetversion	19
3.7.1.6	NXinquirefile	19
3.8	Linking	19
3.8.1	Function Documentation	20
3.8.1.1	NXgetdataID	20
3.8.1.2	NXgetgroupID	20
3.8.1.3	NXmakelink	20
3.8.1.4	NXmakenamedlink	21
3.8.1.5	NXsameID	21

3.9	Memory allocation	21
3.9.1	Function Documentation	21
3.9.1.1	NXfree	21
3.9.1.2	NXmalloc	22
3.10	External linking	22
3.10.1	Function Documentation	22
3.10.1.1	NXisexternaldataset	23
3.10.1.2	NXisexternalgroup	23
3.10.1.3	NXlinkexternal	23
3.10.1.4	NXlinkexternaldataset	24
4	Example Documentation	25
4.1	napi_test.c	25

Chapter 1

NeXus API documentation

2000-2008 NeXus Group

1.1 Purpose of API

The NeXus Application Program Interface is a suite of subroutines, written in C but with wrappers in C++, JAVA, PYTHON, Fortran 77 and 90. The subroutines call HDF routines to read and write the NeXus files with the correct structure.

An API serves a number of useful purposes:

- It simplifies the reading and writing of NeXus files.
- It ensures a certain degree of compliance with the NeXus standard.
- It allows the development of sophisticated input/output features such as automatic unit conversion. This has not been implemented yet.
- It hides the implementation details of the format. In particular, the API can read and write HDF4, HDF5 (and shortly XML) files using the same routines. For these reasons, we request that all NeXus files are written using the supplied API. We cannot be sure that anything written using the underlying HDF API will be recognized by NeXus-aware utilities.

1.2 Core API

The core API provides the basic routines for reading, writing and navigating NeXus files. It is designed to be modal; there is a hidden state that determines which groups and data sets are open at any given moment, and subsequent operations are implicitly performed on these entities. This cuts down the number of parameters to pass around in API calls, at the cost of forcing a certain pre-approved mode d'emploi. This mode d'emploi will be familiar to most: it is very similar to navigating a directory hierarchy;

in our case, NeXus groups are the directories, which contain data sets and/or other directories.

The core API comprises several functional groups which are listed on the **Modules** tab.

C programs that call the above routines should include the following header file:

```
#include "napi.h"
```

See also

napi_test.c

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

C API	5
Data Types	5
General Initialisation and shutdown	5
Reading and Writing Groups	7
Reading and Writing Data	8
General File navigation	14
Meta data routines	17
Linking	19
Memory allocation	21
External linking	22

Chapter 3

Module Documentation

3.1 C API

Modules

- **Data Types**
- **General Initialisation and shutdown**
- **Reading and Writing Groups**
- **Reading and Writing Data**
- **General File navigation**
- **Meta data routines**
- **Linking**
- **Memory allocation**
- **External linking**

3.2 Data Types

Defines

- #define **NX_FLOAT32** 5
32 bit float

3.3 General Initialisation and shutdown

Functions

- NXstatus **NXopen** (CONSTCHAR *filename, NXaccess access_method, NX-handle *pHandle)
Open a NeXus file.

- NXstatus **NXreopen** (NXhandle pOrigHandle, NXhandle *pNewHandle)
Opens an existing NeXus file a second time for e.g.
- NXstatus **NXclose** (NXhandle *pHandle)
close a NeXus file
- NXstatus **NXsetcache** (long newVal)
A function for setting the default cache size for HDF-5.

3.3.1 Function Documentation

3.3.1.1 NXstatus NXclose (NXhandle * *pHandle*)

close a NeXus file

Parameters

<i>pHandle</i>	A NeXus file handle as returned from NXopen. pHandle is invalid after this call.
----------------	--

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.3.1.2 NXstatus NXopen (CONSTCHAR * *filename*, NXaccess *access_method*, NXhandle * *pHandle*)

Open a NeXus file.

NXopen honours full path file names. But it also searches for files in all the paths given in the NX_LOAD_PATH environment variable. NX_LOAD_PATH is supposed to hold a list of path string separated by the platform specific path separator. For unix this is the : , for DOS the ; . Please note that crashing on an open NeXus file will result in corrupted data. Only after a NXclose or a NXflush will the data file be valid.

Parameters

<i>filename</i>	The name of the file to open
<i>access_ - method</i>	The file access method. This can be: <ul style="list-style-type: none"> • NXACC_READ read access • NXACC_RDWR read write access • NXACC_CREATE, NXACC_CREATE4 create a new HDF-4 NeXus file • NXACC_CREATE5 create a new HDF-5 NeXus file • NXACC_CREATEXML create an XML NeXus file. see #NXaccess_ - mode Support for HDF-4 is deprecated.
<i>pHandle</i>	A file handle which will be initialized upon successful completion of NXopen.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.3.1.3 NXstatus NXreopen (NXhandle *pOrigHandle*, NXhandle * *pNewHandle*)

Opens an existing NeXus file a second time for e.g. access from another thread.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.4 Reading and Writing Groups

Functions

- NXstatus **NXmakegroup** (NXhandle *handle*, CONSTCHAR **name*, CONSTCHAR **NXclass*)
NeXus groups are NeXus way of structuring information into a hierarchy.
- NXstatus **NXopengroup** (NXhandle *handle*, CONSTCHAR **name*, CONSTCHAR **NXclass*)
Step into a group.
- NXstatus **NXclosegroup** (NXhandle *handle*)
Closes the currently open group and steps one step down in the NeXus file hierarchy.

3.4.1 Function Documentation

3.4.1.1 NXstatus NXclosegroup (NXhandle *handle*)

Closes the currently open group and steps one step down in the NeXus file hierarchy.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.4.1.2 NXstatus NXmakegroup (NXhandle *handle*, CONSTCHAR * *name*, CONSTCHAR * *NXclass*)

NeXus groups are NeXus way of structuring information into a hierarchy.

This function creates a group but does not open it.

Parameters

<i>handle</i>	A NeXus file handle as initialized NXopen.
<i>name</i>	The name of the group
<i>NXclass</i>	The class name of the group. Should start with the prefix NX

Generated by Doxygen 1.8.17 for NeXus 5.12.0 on 2018-08-27. For details on the format see <http://www.doxygen.org>

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.4.1.3 NXstatus NXopengroup (NXhandle *handle*, CONSTCHAR * *name*, CONSTCHAR * *NXclass*)

Step into a group.

All further access will be within the opened group.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>name</i>	The name of the group
<i>NXclass</i>	the class name of the group. Should start with the prefix NX

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5 Reading and Writing Data

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, const void *data)
Write data to a dataset which has previously been opened with NXopendata.
- NXstatus **NXputattr** (NXhandle handle, CONSTCHAR *name, const void *data, int iDataLen, int iType)
Write an attribute.
- NXstatus **NXputslab** (NXhandle handle, const void *data, const int start[], const 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, const int start[], const 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.

3.5.1 Function Documentation

3.5.1.1 NXstatus NXclosedata (NXhandle handle)

Close access to a dataset.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
---------------	---

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.2 NXstatus NXcompakedata (NXhandle handle, CONSTCHAR * label, int datatype, int rank, int dim[], int comp_typ, int bufsize[])

Create a compressed dataset.

The dataset is NOT opened. Data from this set will automatically be compressed when writing and decompressed on reading.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>label</i>	The name of the dataset
<i>datatype</i>	The data type of this data set.
<i>rank</i>	The number of dimensions this dataset is going to have

<i>comp_type</i>	The compression scheme to use. Possible values: <ul style="list-style-type: none"> • NX_COMP_NONE no compression • NX_COMP_LZW lossless Lempel Ziv Welch compression (recommended) • NX_COMP_RLE run length encoding (only HDF-4) • NX_COMP_HUF Huffmann encoding (only HDF-4)
<i>dim</i>	An array of size rank holding the size of the dataset in each dimension. The first dimension can be NX_UNLIMITED. Data can be appended to such a dimension using NXputslab.
<i>bufsize</i>	The dimensions of the subset of the data which usually be written in one go. This is a parameter used by HDF for performance optimisations. If you write your data in one go, this should be the same as the data dimension. If you write it in slabs, this is your preferred slab size.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.3 NXstatus NXcompress (NXhandle *handle*, int *compr_type*)

Switch compression on.

This routine is superseded by NXcompmakedata and thus is deprecated.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>compr_type</i>	The compression scheme to use. Possible values: <ul style="list-style-type: none"> • NX_COMP_NONE no compression • NX_COMP_LZW lossless Lempel Ziv Welch compression (recommended) • NX_COMP_RLE run length encoding (only HDF-4) • NX_COMP_HUF Huffmann encoding (only HDF-4)

3.5.1.4 NXstatus NXflush (NXhandle * *pHandle*)

flush data to disk

Parameters

<i>pHandle</i>	A NeXus file handle as initialized by NXopen.
----------------	---

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.5 NXstatus NXgetattr (NXhandle *handle*, char * *name*, void * *data*, int * *iDataLen*, int * *iType*)

Read an attribute.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>name</i>	The name of the attribute to read.
<i>data</i>	A pointer to a memory area large enough to hold the attributes value.
<i>iDataLen</i>	The length of data in bytes.
<i>iType</i>	A pointer to an integer which will had been set to the NeXus data type of the attribute.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.6 NXstatus NXgetdata (NXhandle *handle*, void * *data*)

Read a complete dataset from the currently open dataset into memory.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>data</i>	A pointer to the memory area where to read the data, too. Data must point to a memory area large enough to accomodate the data read. Otherwise your program may behave in unexpected and unwelcome ways.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.7 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.

In order to search attributes multiple calls to **NXgetnextattr** (p. 11) are performed in a loop until **NXgetnextattr** (p. 11) returns NX_EOD which indicates that there are no further attributes. reset search using **NXinitattdir** (p. 15)

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>pName</i>	The name of the attribute
<i>iLength</i>	A pointer to an integer which be set to the length of the attribute data.
<i>iType</i>	A pointer to an integer which be set to the NeXus data type of the attribute.

Returns

NX_OK on success, NX_ERROR in the case of an error, NX_EOD when there are no more items.

3.5.1.8 NXstatus NXgetslab (NXhandle *handle*, void * *data*, const int *start*[], const int *size*[])

Read a subset of data from file into memory.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>data</i>	A pointer to the memory data where to copy the data too. The pointer must point to a memory area large enough to accomodate the size of the data read.
<i>start</i>	An array holding the start indices where to start reading the data subset.
<i>size</i>	An array holding the size of the data subset to read for each dimension.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.9 NXstatus NXmakedata (NXhandle *handle*, CONSTCHAR * *label*, int *datatype*, int *rank*, int *dim*[])

Create a multi dimensional data array or dataset.

The dataset is NOT opened.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>label</i>	The name of the dataset
<i>datatype</i>	The data type of this data set.
<i>rank</i>	The number of dimensions this dataset is going to have
<i>dim</i>	An array of size rank holding the size of the dataset in each dimension. The first dimension can be NX_UNLIMITED. Data can be appended to such a dimension using NXputslab.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.10 NXstatus NXopendata (NXhandle *handle*, CONSTCHAR * *label*)

Open access to a dataset.

After this call it is possible to write and read data or attributes to and from the dataset.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>label</i>	The name of the dataset

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.11 NXstatus NXputattr (NXhandle *handle*, CONSTCHAR * *name*, const void * *data*, int *iDataLen*, int *iType*)

Write an attribute.

The kind of attribute written depends on the position in the `file`: at root level, a global attribute is written, if a group is open but no dataset, a group attribute is written, if a dataset is open, a dataset attribute is written.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>name</i>	The name of the attribute.
<i>data</i>	A pointer to the data to write for the attribute.
<i>iDataLen</i>	The length of the data in data in bytes.
<i>iType</i>	The NeXus data type of the attribute.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.12 NXstatus NXputdata (NXhandle *handle*, const void * *data*)

Write data to a dataset which has previously been opened with NXopendata.

This writes all the data in one go. Data should be a pointer to a memory area matching the datatype and dimensions of the dataset.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>data</i>	Pointer to data to write.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.13 NXstatus NXputslab (NXhandle *handle*, const void * *data*, const int *start*[], const int *size*[])

Write a subset of a multi dimensional dataset.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>data</i>	A pointer to a memory area holding the data to write.
<i>start</i>	An array holding the start indices where to start the data subset.
<i>size</i>	An array holding the size of the data subset to write in each dimension.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.5.1.14 NXstatus NXsetnumberformat (NXhandle *handle*, int *type*, char * *format*)

Sets the format for number printing.

This call has only an effect when using the XML physical file format.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>type</i>	The NeXus data type to set the format for.
<i>format</i>	The C-language format string to use for this data type.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.6 General File navigation

Functions

- NXstatus **NXopenpath** (NXhandle handle, CONSTCHAR *path)
Open the NeXus object with the path specified.
- NXstatus **NXopengrouppath** (NXhandle handle, CONSTCHAR *path)
Opens the group in which the NeXus object with the specified path exists.
- NXstatus **NXgetpath** (NXhandle handle, char *path, int pathlen)
Retrieve the current path in the NeXus file.
- NXstatus **NXopensourcegroup** (NXhandle handle)
Open the source group of a linked group or dataset.
- NXstatus **NXgetnextentry** (NXhandle handle, NXname name, NXname nxclass, int *datatype)
Get the next entry in the currently open group.
- NXstatus **NXinitgroupdir** (NXhandle handle)
Resets a pending group search to the start again.
- NXstatus **NXinitattrdir** (NXhandle handle)
Resets a pending attribute search to the start again.

3.6.1 Function Documentation

3.6.1.1 NXstatus NXgetnextentry (NXhandle *handle*, NXname *name*, NXname *nxclass*, int * *datatype*)

Get the next entry in the currently open group.

This is for retrieving information about the content of a NeXus group. In order to search a group **NXgetnextentry** (p. 15) is called in a loop until **NXgetnextentry** (p. 15) returns NX_EOD which indicates that there are no further items in the group. Reset search using **NXinitgroupdir** (p. 16)

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>name</i>	The name of the object
<i>nxclass</i>	The NeXus class name for a group or the string SDS for a dataset.
<i>datatype</i>	The NeXus data type if the item is a SDS.

Returns

NX_OK on success, NX_ERROR in the case of an error, NX_EOD when there are no more items.

3.6.1.2 NXstatus NXgetpath (NXhandle *handle*, char * *path*, int *pathlen*)

Retrieve the current path in the NeXus file.

Parameters

<i>handle</i>	a NeXus file handle
<i>path</i>	A buffer to copy the path too
<i>pathlen</i>	The maximum number of characters to copy into path

Returns

NX_OK or NX_ERROR

3.6.1.3 NXstatus NXinitattrdir (NXhandle *handle*)

Resets a pending attribute search to the start again.

To be called in a **NXgetnextattr** (p. 11) loop when an attribute search has to be restarted.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
---------------	---

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.6.1.4 NXstatus NXinitgroupdir (NXhandle *handle*)

Resets a pending group search to the start again.

To be called in a **NXgetnextentry** (p. 15) loop when a group search has to be restarted.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
---------------	---

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.6.1.5 NXstatus NXopengrouppath (NXhandle *handle*, CONSTCHAR * *path*)

Opens the group in which the NeXus object with the specified path exists.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>path</i>	A unix like path string to a NeXus group or dataset. The path string is a list of group names and SDS names separated with / (slash). Example: /entry1/sample/name

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.6.1.6 NXstatus NXopenpath (NXhandle *handle*, CONSTCHAR * *path*)

Open the NeXus object with the path specified.

Parameters

<i>handle</i>	A NeXus file handle as returned from NXopen.
<i>path</i>	A unix like path string to a NeXus group or dataset. The path string is a list of group names and SDS names separated with / (slash). Example: /entry1/sample/name

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.6.1.7 NXstatus NXopensourcegroup (NXhandle *handle*)

Open the source group of a linked group or dataset.

Returns an error when the item is not a linked item.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
---------------	---

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.7 Meta data routines

Functions

- NXstatus **NXgetinfo** (NXhandle handle, int *rank, int dimension[], int *datatype)

Retrieve information about the currently open dataset.

- NXstatus **NXgetattrinfo** (NXhandle handle, int *no_items)

Get the count of attributes in the currently open dataset, group or global attributes when at root level.

- NXstatus **NXgetgroupinfo** (NXhandle handle, int *no_items, NXname name, NXname nxclass)

Retrieve information about the currently open group.

- NXstatus **NXinquirefile** (NXhandle handle, char *filename, int filenameBufferLength)

Inquire the filename of the currently open file.

- const char * **NXgetversion** ()

Utility function to return NeXus version.

- NXstatus **NXgetrawinfo** (NXhandle handle, int *rank, int dimension[], int *datatype)

Retrieve information about the currently open dataset.

3.7.1 Function Documentation

3.7.1.1 NXstatus NXgetattrinfo (NXhandle *handle*, int * *no_items*)

Get the count of attributes in the currently open dataset, group or global attributes when at root level.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>no_items</i>	A pointer to an integer which be set to the number of attributes available.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.7.1.2 NXstatus NXgetgroupinfo (NXhandle *handle*, int * *no_items*, NXname *name*, NXname *nxclass*)

Retrieve information about the currently open group.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>no_items</i>	A pointer to an integer which will be set to the count of group elements available. This is the count of other groups and data sets in this group.
<i>name</i>	The name of the group.
<i>nxclass</i>	The NeXus class name of the group.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.7.1.3 NXstatus NXgetinfo (NXhandle *handle*, int * *rank*, int *dimension*[], int * *datatype*)

Retrieve information about the currently open dataset.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>rank</i>	A pointer to an integer which will be filled with the rank of the dataset.
<i>dimension</i>	An array which will be initialized with the size of the dataset in any of its dimensions. The array must have at least the size of rank.
<i>datatype</i>	A pointer to an integer which will be set to the NeXus data type code for this dataset.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.7.1.4 NXstatus NXgetrawinfo (NXhandle *handle*, int * *rank*, int *dimension*[], int * *datatype*)

Retrieve information about the currently open dataset.

In contrast to the main function below, this function does not try to find out about the size of strings properly.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>rank</i>	A pointer to an integer which will be filled with the rank of the dataset.
<i>dimension</i>	An array which will be initialized with the size of the dataset in any of its dimensions. The array must have at least the size of rank.
<i>datatype</i>	A pointer to an integer which will be set to the NeXus data type code for this dataset.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.7.1.5 `const char* NXgetversion ()`

Utility function to return NeXus version.

Returns

pointer to string in static storage. Version in same format as NEXUS_VERSION string in napi.h i.e. "major.minor.patch"

3.7.1.6 `NXstatus NXinquirefile (NXhandle handle, char * filename, int filenameBufferLength)`

Inquire the filename of the currently open file.

filenameBufferLength of the file name will be copied into the filename buffer.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>filename</i>	The buffer to hold the filename.
<i>filename-BufferLength</i>	The length of the filename buffer.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.8 Linking

Functions

- NXstatus **NXgetdataID** (NXhandle handle, NXlink *pLink)
Retrieve link data for a dataset.
- NXstatus **NXmakelink** (NXhandle handle, NXlink *pLink)
Create a link to the group or dataset described by pLink in the currently open group.
- NXstatus **NXmakenamedlink** (NXhandle handle, CONSTCHAR *newname, NXlink *pLink)
Create a link to the group or dataset described by pLink in the currently open group.
- NXstatus **NXgetgroupID** (NXhandle handle, NXlink *pLink)
Retrieve link data for the currently open group.
- NXstatus **NXsameID** (NXhandle handle, NXlink *pFirstID, NXlink *pSecondID)
Tests if two link data structures describe the same item.

3.8.1 Function Documentation

3.8.1.1 NXstatus NXgetdataID (NXhandle *handle*, NXlink * *pLink*)

Retrieve link data for a dataset.

This link data can later on be used to link this dataset into a different group.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>pLink</i>	A link data structure which will be initialized with the required information for linking.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.8.1.2 NXstatus NXgetgroupID (NXhandle *handle*, NXlink * *pLink*)

Retrieve link data for the currently open group.

This link data can later on be used to link this group into a different group.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>pLink</i>	A link data structure which will be initialized with the required information for linking.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.8.1.3 NXstatus NXmakelink (NXhandle *handle*, NXlink * *pLink*)

Create a link to the group or dataset described by pLink in the currently open group.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>pLink</i>	A link data structure describing the object to link. This must have been initialized by either a call to NXgetdataID or NXgetgroupID.

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.8.1.4 `NXstatus NXmakenamedlink (NXhandle handle, CONSTCHAR * newname, NXlink * pLink)`

Create a link to the group or dataset described by *pLink* in the currently open group.
But give the linked item a new name.

Parameters

<i>handle</i>	A NeXus file handle as initialized by <code>NXopen</code> .
<i>newname</i>	The new name of the item in the currently open group.
<i>pLink</i>	A link data structure describing the object to link. This must have been initialized by either a call to <code>NXgetdataID</code> or <code>NXgetgroupID</code> .

Returns

`NX_OK` on success, `NX_ERROR` in the case of an error.

3.8.1.5 `NXstatus NXsamelD (NXhandle handle, NXlink * pFirstID, NXlink * pSecondID)`

Tests if two link data structures describe the same item.

Parameters

<i>handle</i>	A NeXus file handle as initialized by <code>NXopen</code> .
<i>pFirstID</i>	The first link data for the test.
<i>pSecondID</i>	The second link data structure.

Returns

`NX_OK` when both link data structures describe the same item, `NX_ERROR` else.

3.9 Memory allocation

Functions

- `NXstatus NXmalloc (void **data, int rank, const int dimensions[], int datatype)`
Utility function which allocates a suitably sized memory area for the dataset characteristics specified.
- `NXstatus NXfree (void **data)`
Utility function to release the memory for data.

3.9.1 Function Documentation

3.9.1.1 `NXstatus NXfree (void ** data)`

Utility function to release the memory for data.

Parameters

<i>data</i>	A pointer to a pointer to free.
-------------	---------------------------------

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.9.1.2 NXstatus NXmalloc (void ** data, int rank, const int dimensions[], int datatype)

Utility function which allocates a suitably sized memory area for the dataset characteristics specified.

Parameters

<i>data</i>	A pointer to a pointer which will be initialized with a pointer to a suitably sized memory area.
<i>rank</i>	the rank of the data.
<i>dimensions</i>	An array holding the size of the data in each dimension.
<i>datatype</i>	The NeXus data type of the data.

Returns

NX_OK when allocation succeeds, NX_ERROR in the case of an error.

3.10 External linking

Functions

- NXstatus **NXisexternalgroup** (NXhandle handle, CONSTCHAR *name, CONSTCHAR *nxclass, char *url, int urlLen)
Test if a group is actually pointing to an external file.
- NXstatus **NXisexternaldataset** (NXhandle handle, CONSTCHAR *name, char *url, int urlLen)
Test if a dataset is actually pointing to an external file.
- NXstatus **NXlinkexternal** (NXhandle handle, CONSTCHAR *name, CONSTCHAR *nxclass, CONSTCHAR *url)
Create a link to a group in an external file.
- NXstatus **NXlinkexternaldataset** (NXhandle handle, CONSTCHAR *name, CONSTCHAR *url)
Create a link to a dataset in an external file.

3.10.1 Function Documentation

3.10.1.1 NXstatus NXisexternaldataset (NXhandle *handle*, CONSTCHAR * *name*, char * *url*, int *urlLen*)

Test if a dataset is actually pointing to an external file.

If so, retrieve the URL of the external file.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>name</i>	The name of the dataset to test.
<i>url</i>	A buffer to copy the URL too.
<i>urlLen</i>	The length of the Url buffer. At maximum urlLen bytes will be copied to url.

Returns

NX_OK when the dataset is pointing to an external file, NX_ERROR else.

3.10.1.2 NXstatus NXisexternalgroup (NXhandle *handle*, CONSTCHAR * *name*, CONSTCHAR * *nxclass*, char * *url*, int *urlLen*)

Test if a group is actually pointing to an external file.

If so, retrieve the URL of the external file.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>name</i>	The name of the group to test.
<i>nxclass</i>	The class name of the group to test.
<i>url</i>	A buffer to copy the URL too.
<i>urlLen</i>	The length of the Url buffer. At maximum urlLen bytes will be copied to url.

Returns

NX_OK when the group is pointing to an external file, NX_ERROR else.

3.10.1.3 NXstatus NXlinkexternal (NXhandle *handle*, CONSTCHAR * *name*, CONSTCHAR * *nxclass*, CONSTCHAR * *url*)

Create a link to a group in an external file.

This works by creating a NeXus group under the current level in the hierarchy which actually points to a group in another file.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>name</i>	The name of the group which points to the external file.
<i>nxclass</i>	The class name of the group which points to the external file.

<i>url</i>	The URL of the external file. Currently only one URL format is supported: <code>nxfile://path-tofile#path-in-file</code> . This consists of two parts: the first part is of course the path to the file. The second part, <code>path-in-file</code> , is the path to the group in the external file which appears in the first file.
------------	--

Returns

NX_OK on success, NX_ERROR in the case of an error.

3.10.1.4 NXstatus NXlinkexternaldataset (NXhandle *handle*, CONSTCHAR * *name*, CONSTCHAR * *url*)

Create a link to a dataset in an external file.

This works by creating a dataset under the current level in the hierarchy which actually points to a dataset in another file.

Parameters

<i>handle</i>	A NeXus file handle as initialized by NXopen.
<i>name</i>	The name of the dataset which points to the external file.
<i>url</i>	The URL of the external file. Currently only one URL format is supported: <code>nxfile://path-tofile#path-in-file</code> . This consists of two parts: the first part is of course the path to the file. The second part, <code>path-in-file</code> , is the path to the dataset in the external file which appears in the first file.

Returns

NX_OK on success, NX_ERROR in the case of an error.

Chapter 4

Example Documentation

4.1 napi_test.c

This is the test program for the NeXus C API. It illustrates calling most functions to read and write a file.

```
/*-----  
  NeXus - Neutron & X-ray Common Data Format  
  
  Test program for C API  
  
  Copyright (C) 1997-2011 Freddie Akeroyd  
  
  This library is free software; you can redistribute it and/or  
  modify it under the terms of the GNU Lesser General Public  
  License as published by the Free Software Foundation; either  
  version 2 of the License, or (at your option) any later version.  
  
  This library is distributed in the hope that it will be useful,  
  but WITHOUT ANY WARRANTY; without even the implied warranty of  
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU  
  Lesser General Public License for more details.  
  
  You should have received a copy of the GNU Lesser General Public  
  License along with this library; if not, write to the Free Software  
  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA  
  
  For further information, see <http://www.nexusformat.org>  
  
  $Id: napi_test.c 1799 2012-01-16 10:10:52Z Freddie Akeroyd $  
-----*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#ifndef _WIN32  
#include <unistd.h>  
#endif  
#include "napi.h"  
#include "napiconfig.h"
```

```

static void print_data (const char *prefix, void *data, int type, int num);
static int testLoadPath();
static int testExternal(char *progName);

static const char *relativePathOf(const char* filename) {
    char cwd[1024];

    getcwd(cwd, sizeof(cwd));

    if (strncmp(filename, cwd, strlen(cwd)) == 0)
    {
        return filename+strlen(cwd)+1;
    }
    else
    {
        return filename;
    }
}

int main (int argc, char *argv[])
{
    int i, j, k, n, NXrank, NXdims[32], NXtype, NXlen, entry_status, attr_status;
    float r;
    void *data_buffer;
    unsigned char i1_array[4] = {1, 2, 3, 4};
    short int i2_array[4] = {1000, 2000, 3000, 4000};
    int i4_array[4] = {1000000, 2000000, 3000000, 4000000};
    float r4_array[5][4] =
    {{1., 2., 3., 4.}, {5., 6., 7., 8.}, {9., 10., 11., 12.}, {13., 14., 15., 16.},
     {17., 18., 19., 20.}};
    double r8_array[5][4] =
    {{1., 2., 3., 4.}, {5., 6., 7., 8.}, {9., 10., 11., 12.}, {13., 14., 15., 16.},
     {17., 18., 19., 20.}};
    int array_dims[2] = {5, 4};
    int unlimited_dims[1] = {NX_UNLIMITED};
    int chunk_size[2]={5,4};
    int slab_start[2], slab_size[2];
    char name[64], char_class[64], char_buffer[128];
    char group_name[64], class_name[64];
    char cl_array[5][4] = {{'a', 'b', 'c', 'd'}, {'e', 'f', 'g', 'h'},
        {'i', 'j', 'k', 'l'}, {'m', 'n', 'o', 'p'}, {'q', 'r', 's', 't'}};
    int unlimited_cdims[2] = {NX_UNLIMITED, 4};
    NXhandle fileid, clone_fileid;
    NXlink glink, dlink, blink;
    int comp_array[100][20];
    int dims[2];
    int cdims[2];
    int nx_creation_code;
    char nxFile[80];
    char filename[256];
    int64_t grossezahl[4];
    const char* ch_test_data = "NeXus >< &{'\\&\" Data";
    char path[512];

    grossezahl[0] = 12;
    grossezahl[2] = 23;
    #if HAVE_LONG_LONG_INT
    grossezahl[1] = (int64_t)555555555555LL;
    grossezahl[3] = (int64_t)777777777777LL;
    #else
    grossezahl[1] = (int64_t)555555555555;
    grossezahl[3] = (int64_t)777777777777;
    #endif

```

```

#endif /* HAVE_LONG_LONG_INT */

if(strstr(argv[0],"napi_test-hdf5") != NULL){
    nx_creation_code = NXACC_CREATE5;
    strcpy(nxFile,"NXtest.h5");
}else if(strstr(argv[0],"napi_test-xml-table") != NULL){
    nx_creation_code = NXACC_CREATEXML | NXACC_TABLE;
    strcpy(nxFile,"NXtest-table.xml");
}else if(strstr(argv[0],"napi_test-xml") != NULL){
    nx_creation_code = NXACC_CREATEXML;
    strcpy(nxFile,"NXtest.xml");
} else {
    nx_creation_code = NXACC_CREATE;
    strcpy(nxFile,"NXtest.hdf");
}

/* create file */
if (NXopen (nxFile, nx_creation_code, &fileid) != NX_OK) return 1;
if (nx_creation_code == NXACC_CREATE5)
{
    if (NXreopen (fileid, &clone_fileid) != NX_OK) return 1;
}
NXsetnumberformat (fileid,NX_FLOAT32,"%9.3f");
if (NXmakegroup (fileid, "entry", "NXentry") != NX_OK) return 1;
if (NXopengroup (fileid, "entry", "NXentry") != NX_OK) return 1;
if (NXputattr(fileid,"hugo","namenlos",strlen("namenlos"), NX_CHAR) != NX_OK) re
    turn 1;
if(NXputattr(fileid,"cucumber","passion",strlen("passion"), NX_CHAR) != NX_OK)
    return 1;
NXlen = strlen(ch_test_data);
if (NXmakedata (fileid, "ch_data", NX_CHAR, 1, &NXlen) != NX_OK) return 1;
if (NXopendata (fileid, "ch_data") != NX_OK) return 1;
    if (NXputdata (fileid, ch_test_data) != NX_OK) return 1;
if (NXclosedata (fileid) != NX_OK) return 1;
if (NXmakedata (fileid, "c1_data", NX_CHAR, 2, array_dims) != NX_OK) return
    1;
if (NXopendata (fileid, "c1_data") != NX_OK) return 1;
    if (NXputdata (fileid, c1_array) != NX_OK) return 1;
if (NXclosedata (fileid) != NX_OK) return 1;
if (NXmakedata (fileid, "i1_data", NX_INT8, 1, &array_dims[1]) != NX_OK) re
    turn 1;
if (NXopendata (fileid, "i1_data") != NX_OK) return 1;
    if (NXputdata (fileid, i1_array) != NX_OK) return 1;
if (NXclosedata (fileid) != NX_OK) return 1;
if (NXmakedata (fileid, "i2_data", NX_INT16, 1, &array_dims[1]) != NX_OK) re
    turn 1;
if (NXopendata (fileid, "i2_data") != NX_OK) return 1;
    if (NXputdata (fileid, i2_array) != NX_OK) return 1;
if (NXclosedata (fileid) != NX_OK) return 1;
if (NXmakedata (fileid, "i4_data", NX_INT32, 1, &array_dims[1]) != NX_OK) re
    turn 1;
if (NXopendata (fileid, "i4_data") != NX_OK) return 1;
    if (NXputdata (fileid, i4_array) != NX_OK) return 1;
if (NXclosedata (fileid) != NX_OK) return 1;
if (NXcompmakedata (fileid, "r4_data", NX_FLOAT32, 2, array_dims,NX_COMP_LZW
    ,chunk_size) != NX_OK) return 1;
if (NXopendata (fileid, "r4_data") != NX_OK) return 1;
    if (NXputdata (fileid, r4_array) != NX_OK) return 1;
if (NXclosedata (fileid) != NX_OK) return 1;
if (NXmakedata (fileid, "r8_data", NX_FLOAT64, 2, array_dims) != NX_OK) retu
    rn 1;
if (NXopendata (fileid, "r8_data") != NX_OK) return 1;

```

```

    slab_start[0] = 4; slab_start[1] = 0; slab_size[0] = 1; slab_size[1] = 4;

    if (NXputslab (fileid, (double*)r8_array + 16, slab_start, slab_size) !=
NX_OK) return 1;
    slab_start[0] = 0; slab_start[1] = 0; slab_size[0] = 4; slab_size[1] = 4;

    if (NXputslab (fileid, r8_array, slab_start, slab_size) != NX_OK) return
1;
    if (NXputattr (fileid, "ch_attribute", ch_test_data, strlen (ch_test_data
), NX_CHAR) != NX_OK) return 1;
    i = 42;
    if (NXputattr (fileid, "i4_attribute", &i, 1, NX_INT32) != NX_OK) return
1;
    r = 3.14159265;
    if (NXputattr (fileid, "r4_attribute", &r, 1, NX_FLOAT32) != NX_OK) retur
n 1;
    if (NXgetdataID (fileid, &dlink) != NX_OK) return 1;
if (NXclosedata (fileid) != NX_OK) return 1;
dims[0] = 4;
if (nx_creation_code != NXACC_CREATE)
{
    if (NXmakedata (fileid, "grosse_zahl", NX_INT64, 1,dims) == NX_OK) {
        if (NXopendata (fileid, "grosse_zahl") != NX_OK) return 1;
        if (NXputdata (fileid, grossezahl) != NX_OK) return 1;
        if (NXclosedata (fileid) != NX_OK) return 1;
    }
}
if (NXmakegroup (fileid, "data", "NXdata") != NX_OK) return 1;
if (NXopengroup (fileid, "data", "NXdata") != NX_OK) return 1;
if (NXmakelink (fileid, &dlink) != NX_OK) return 1;
dims[0] = 100;
dims[1] = 20;
for(i = 0; i < 100; i++)
{
    for(j = 0; j < 20; j++)
    {
        comp_array[i][j] = i;
    }
}
cdims[0] = 20;
cdims[1] = 20;
if (NXcompmakedata (fileid, "comp_data", NX_INT32, 2, dims, NX_COMP_LZW,
cdims) != NX_OK) return 1;
if (NXopendata (fileid, "comp_data") != NX_OK) return 1;
if (NXputdata (fileid, comp_array) != NX_OK) return 1;
if (NXclosedata (fileid) != NX_OK) return 1;
if (NXflush (&fileid) != NX_OK) return 1;
if (NXmakedata (fileid, "flush_data", NX_INT32, 1, unlimited_dims) != NX_
OK) return 1;
slab_size[0] = 1;
for (i = 0; i < 7; i++)
{
    slab_start[0] = i;
    if (NXopendata (fileid, "flush_data") != NX_OK) return 1;
    if (NXputslab (fileid, &i, slab_start, slab_size) != NX_OK) retur
n 1;
    if (NXflush (&fileid) != NX_OK) return 1;
}
if (NXclosegroup (fileid) != NX_OK) return 1;
if (NXmakegroup (fileid, "sample", "NXsample") != NX_OK) return 1;
if (NXopengroup (fileid, "sample", "NXsample") != NX_OK) return 1;
NXlen = 12;

```

```

    if (NXmakedata (fileid, "ch_data", NX_CHAR, 1, &NXlen) != NX_OK) return 1
;
    if (NXopendata (fileid, "ch_data") != NX_OK) return 1;
    if (NXputdata (fileid, "NeXus sample") != NX_OK) return 1;
    if (NXclosedata (fileid) != NX_OK) return 1;
    if (NXgetgroupID (fileid, &glink) != NX_OK) return 1;
    if (( nx_creation_code & NXACC_CREATEXML) == 0 ) {
        if (NXmakedata (fileid, "cdata_unlimited", NX_CHAR, 2, unlimited_cdim
s) != NX_OK) return 1;
        if (NXopendata (fileid, "cdata_unlimited") != NX_OK) return 1;
        slab_size[0] = 1;
        slab_size[1] = 4;
        slab_start[1] = 0;
        for (i = 0; i < 5; i++)
        {
            slab_start[0] = i;
            if (NXputslab (fileid, &(cl_array[i][0]), slab_start, slab_size) !
= NX_OK) return 1;
        }
        if (NXclosedata (fileid) != NX_OK) return 1;
    }
    if (NXclosegroup (fileid) != NX_OK) return 1;
if (NXclosegroup (fileid) != NX_OK) return 1;
if (NXmakegroup (fileid, "link", "NXentry") != NX_OK) return 1;
if (NXopengroup (fileid, "link", "NXentry") != NX_OK) return 1;
    if (NXmakelink (fileid, &glink) != NX_OK) return 1;
    if (NXmakenamedlink (fileid, "renLinkGroup", &glink) != NX_OK) return 1;
    if (NXmakenamedlink (fileid, "renLinkData", &dlink) != NX_OK) return 1;
if (NXclosegroup (fileid) != NX_OK) return 1;
if (NXclose (&fileid) != NX_OK) return 1;

if ( ( argc >= 2) && !strcmp(argv[1], "-q") )
{
    return 0; /* create only */
}
/*
read test
*/
if (NXopen (nxFile, NXACC_RDWR, &fileid) != NX_OK) return 1;
if (NXinquirefile (fileid, filename, 256) != NX_OK) {
    return 1;
}
printf("NXinquirefile found: %s\n", relativePathOf(filename));
NXgetattrinfo (fileid, &i);
if (i > 0) {
    printf ("Number of global attributes: %d\n", i);
}
do {
    attr_status = NXgetnextattr (fileid, name, NXdims, &NXtype);
    if (attr_status == NX_ERROR) return 1;
    if (attr_status == NX_OK) {
        switch (NXtype) {
            case NX_CHAR:
                NXlen = sizeof (char_buffer);
                if (NXgetattr (fileid, name, char_buffer, &NXlen, &NXtype)
!= NX_OK) return 1;
                if ( strcmp(name, "file_time") &&
                    strcmp(name, "HDF_version") &&
                    strcmp(name, "HDF5_Version") &&
                    strcmp(name, "XML_version") )
                {
                    printf (" %s = %s\n", name, char_buffer);
                }

```



```

        if (NXgetslab (fileid, data_buffer, slab_start, slab_size) != NX
_OK) return 1;
        print_data ("      ", data_buffer, NXtype, 4);
        slab_start[0] = 2;
        if (NXgetslab (fileid, data_buffer, slab_start, slab_size) != NX
_OK) return 1;
        print_data ("      ", data_buffer, NXtype, 4);
        slab_start[0] = 3;
        if (NXgetslab (fileid, data_buffer, slab_start, slab_size) != NX
_OK) return 1;
        print_data ("      ", data_buffer, NXtype, 4);
        slab_start[0] = 4;
        if (NXgetslab (fileid, data_buffer, slab_start, slab_size) != NX
_OK) return 1;
        print_data ("      ", data_buffer, NXtype, 4);
        if (NXgetattrinfo (fileid, &i) != NX_OK) return 1;
        if (i > 0) {
            printf ("      Number of attributes : %d\n", i);
        }
        do {
            attr_status = NXgetnextattr (fileid, name, NXdims, &NXtype);
            if (attr_status == NX_ERROR) return 1;
            if (attr_status == NX_OK) {
                switch (NXtype) {
                    case NX_INT32:
                        NXlen = 1;
                        if (NXgetattr (fileid, name, &i, &NXlen, &NXtype) !=
NX_OK) return 1;
                        printf ("      %s : %d\n", name, i);
                        break;
                    case NX_FLOAT32:
                        NXlen = 1;
                        if (NXgetattr (fileid, name, &r, &NXlen, &NXtype) !=
NX_OK) return 1;
                        printf ("      %s : %f\n", name, r);
                        break;
                    case NX_CHAR:
                        NXlen = sizeof (char_buffer);
                        if (NXgetattr (fileid, name, char_buffer, &NXlen, &N
Xtype) != NX_OK) return 1;
                        printf ("      %s : %s\n", name, char_buffer);
                        break;
                }
            }
        } while (attr_status == NX_OK);
    }
    if (NXclosedata (fileid) != NX_OK) return 1;
    if (NXfree ((void **) &data_buffer) != NX_OK) return 1;
}
}
} while (entry_status == NX_OK);
if (NXclosegroup (fileid) != NX_OK) return 1;
/*
 * check links
 */
if (NXopengroup (fileid, "entry", "NXentry") != NX_OK) return 1;
if (NXopengroup (fileid, "sample", "NXsample") != NX_OK) return 1;
    if (NXgetgroupID (fileid, &glink) != NX_OK) return 1;
if (NXclosegroup (fileid) != NX_OK) return 1;
if (NXopengroup (fileid, "data", "NXdata") != NX_OK) return 1;
    if (NXopendata (fileid, "r8_data") != NX_OK) return 1;

```

```

        if (NXgetdataID (fileid, &dlink) != NX_OK) return 1;
        if (NXclosedata (fileid) != NX_OK) return 1;
    if (NXclosegroup (fileid) != NX_OK) return 1;
    if (NXopendata (fileid, "r8_data") != NX_OK) return 1;
        if (NXgetdataID (fileid, &blink) != NX_OK) return 1;
    if (NXclosedata (fileid) != NX_OK) return 1;
    if (NXsameID(fileid, &dlink, &blink) != NX_OK)
    {
        printf ("Link check FAILED (r8_data)\n");
        printf ("original data\n");
        NXIprintlink(fileid, &dlink);
        printf ("linked data\n");
        NXIprintlink(fileid, &blink);
        return 1;
    }
if (NXclosegroup (fileid) != NX_OK) return 1;

if (NXopengroup (fileid, "link", "NXentry") != NX_OK) return 1;
if (NXopengroup (fileid, "sample", "NXsample") != NX_OK) return 1;
if (NXgetpath(fileid,path,512) != NX_OK) return 1;
printf("Group path %s\n", path);
if (NXgetgroupID (fileid, &blink) != NX_OK) return 1;
    if (NXsameID(fileid, &glink, &blink) != NX_OK)
    {
        printf ("Link check FAILED (sample)\n");
        printf ("original group\n");
        NXIprintlink(fileid, &glink);
        printf ("linked group\n");
        NXIprintlink(fileid, &blink);
        return 1;
    }
if (NXclosegroup (fileid) != NX_OK) return 1;

if (NXopengroup (fileid, "renLinkGroup", "NXsample") != NX_OK) return 1;
if (NXgetgroupID (fileid, &blink) != NX_OK) return 1;
    if (NXsameID(fileid, &glink, &blink) != NX_OK)
    {
        printf ("Link check FAILED (renLinkGroup)\n");
        printf ("original group\n");
        NXIprintlink(fileid, &glink);
        printf ("linked group\n");
        NXIprintlink(fileid, &blink);
        return 1;
    }
if (NXclosegroup (fileid) != NX_OK) return 1;

if (NXopendata(fileid,"renLinkData") != NX_OK) return 1;
if (NXgetdataID(fileid,&blink) != NX_OK) return 1;
    if (NXsameID(fileid, &dlink, &blink) != NX_OK)
    {
        printf ("Link check FAILED (renLinkData)\n");
        printf ("original group\n");
        NXIprintlink(fileid, &glink);
        printf ("linked group\n");
        NXIprintlink(fileid, &blink);
        return 1;
    }
if (NXclosedata(fileid) != NX_OK) return 1;
if (NXclosegroup (fileid) != NX_OK) return 1;
printf ("Link check OK\n");

/*

```

```

    tests for NXopenpath
*/
if(NXopenpath(fileid, "/entry/data/comp_data") != NX_OK) {
    printf("Failure on NXopenpath\n");
    return 0;
}
if(NXopenpath(fileid, "/entry/data/comp_data") != NX_OK) {
    printf("Failure on NXopenpath\n");
    return 0;
}
if(NXopenpath(fileid, "../r8_data") != NX_OK) {
    printf("Failure on NXopenpath\n");
    return 0;
}
if(NXopengrouppath(fileid, "/entry/data/comp_data") != NX_OK) {
    printf("Failure on NXopengrouppath\n");
    return 0;
}
if(NXopenpath(fileid, "/entry/data/r8_data") != NX_OK) {
    printf("Failure on NXopenpath\n");
    return 0;
}
printf("NXopenpath checks OK\n");

if (NXclose (&fileid) != NX_OK) return 1;

if(testLoadPath() != 0) return 1;

if(testExternal(argv[0]) != 0) {
    return 1;
}

return 0;
}
/*-----*/
static int testLoadPath() {
    NXhandle h;

    if(getenv("NX_LOAD_PATH") != NULL) {
        if (NXopen ("dmc01.hdf", NXACC_RDWR, &h) != NX_OK) {
            printf("Loading NeXus file dmc01.hdf from path %s FAILED\n", getenv("NX_LOAD_PATH"));
            return 1;
        } else {
            printf("Success loading NeXus file from path\n");
            NXclose(&h);
            return 0;
        }
    }
    return 0;
}
/*-----*/
static int testExternal(char *progName) {
    char nxfile[255], ext[5], testFile[80], time[132], filename[256];
    int create;
    NXhandle hfil;
    int dummylen = 1;
    float dummyfloat = 1;
    float temperature;

    if(strstr(progName, "hdf4") != NULL) {
        strcpy(ext, "hdf");
    }
}

```

```

    create = NXACC_CREATE;
} else if (strstr(progName, "hdf5") != NULL) {
    strcpy(ext, "h5");
    create = NXACC_CREATE5;
} else if (strstr(progName, "xml") != NULL) {
    strcpy(ext, "xml");
    create = NXACC_CREATEXML;
} else {
    printf("Failed to recognise napi_test program in testExternal\n");
    return 1;
}

sprintf(testFile, "nnext.%s", ext);

/*
 * create the test file
 */
if (NXopen(testFile, create, &hfil) != NX_OK) {
    return 1;
}
/*if (NXmakegroup(hfil, "entry1", "NXentry") != NX_OK) {
    return 1;
}*/
sprintf(nxfile, "nxfile://data/dmc01.%s#/entry1", ext);
if (NXlinkexternal(hfil, "entry1", "NXentry", nxfile) != NX_OK) {
    return 1;
}
/*if (NXmakegroup(hfil, "entry2", "NXentry") != NX_OK) {
    return 1;
}*/
sprintf(nxfile, "nxfile://data/dmc02.%s#/entry1", ext);
if (NXlinkexternal(hfil, "entry2", "NXentry", nxfile) != NX_OK) {
    return 1;
}
if (NXmakegroup(hfil, "entry3", "NXentry") != NX_OK) {
    return 1;
}
if (NXopengroup(hfil, "entry3", "NXentry") != NX_OK) {
    return 1;
}
if (NXmakedata(hfil, "extlinkdata", NX_FLOAT32, 1, &dummylen) != NX_OK) return
    1;
if (NXopendata(hfil, "extlinkdata") != NX_OK) return 1;
if (NXputdata(hfil, &dummyfloat) != NX_OK) return 1;
sprintf(nxfile, "nxfile://data/dmc01.%s#/entry1/sample/temperature_mean", ext);
if (NXputattr(hfil, "napimount", nxfile, strlen(nxfile), NX_CHAR) != NX_OK) return
    1;
if (NXclose(&hfil) != NX_OK) {
    return 1;
}

/*
 * actually test linking
 */
if (NXopen(testFile, NXACC_RDWR, &hfil) != NX_OK) {
    return 1;
}
if (NXopenpath(hfil, "/entry1/start_time") != NX_OK) {
    return 1;
}
memset(time, 0, 132);
if (NXgetdata(hfil, time) != NX_OK) {

```

```

    return 1;
}
printf("First file time: %s\n", time);

if(NXInquirefile(hfil,filename,256) != NX_OK){
    return 1;
}
printf("NXInquirefile found: %s\n", relativePathOf(filename));

if(NXOpenpath(hfil,"/entry2/sample/sample_name") != NX_OK){
    return 1;
}
memset(time,0,132);
if(NXgetdata(hfil,time) != NX_OK){
    return 1;
}
printf("Second file sample: %s\n", time);
if(NXInquirefile(hfil,filename,256) != NX_OK){
    return 1;
}
printf("NXInquirefile found: %s\n", relativePathOf(filename));

if(NXOpenpath(hfil,"/entry2/start_time") != NX_OK){
    return 1;
}
memset(time,0,132);
if(NXgetdata(hfil,time) != NX_OK){
    return 1;
}
printf("Second file time: %s\n", time);
NXOpenpath(hfil,"/");
if(NXisexternalgroup(hfil,"entry1","NXentry",filename,255) != NX_OK){
    return 1;
} else {
    printf("entry1 external URL = %s\n", filename);
}

printf("testing link to external data set\n");
if(NXOpenpath(hfil,"/entry3") != NX_OK){
    return 1;
}
if (NXopendata (hfil, "extlinkdata") != NX_OK) return 1;
memset (&temperature,0,4);
if(NXgetdata(hfil,&temperature) != NX_OK){
    return 1;
}
printf("value retrieved: %4.2f\n", temperature);

NXclose(&hfil);
printf("External File Linking tested OK\n");
return 0;
}
/*-----*/
static void
print_data (const char *prefix, void *data, int type, int num)
{
    int i;
    printf ("%s", prefix);
    for (i = 0; i < num; i++) {
        switch (type) {
            case NX_CHAR:
                printf ("%c", ((char *) data)[i]);

```

```
        break;

    case NX_INT8:
        printf (" %d", ((unsigned char *) data)[i]);
        break;

    case NX_INT16:
        printf (" %d", ((short *) data)[i]);
        break;

    case NX_INT32:
        printf (" %d", ((int *) data)[i]);
        break;

    case NX_INT64:
        printf (" %lld", (long long)((int64_t *) data)[i]);
        break;

    case NX_UINT64:
        printf (" %llu", (unsigned long long)((uint64_t *) data)[i]);
        break;

    case NX_FLOAT32:
        printf (" %f", ((float *) data)[i]);
        break;

    case NX_FLOAT64:
        printf (" %f", ((double *) data)[i]);
        break;

    default:
        printf ("print_data: invalid type");
        break;
    }
}
printf ("\n");
}
```

Index

- C API, 5
- Data Types, 5
- External linking, 22
 - NXisexternaldataset, 22
 - NXisexternalgroup, 23
 - NXlinkexternal, 23
 - NXlinkexternaldataset, 24
- General File navigation, 14
 - NXgetnextentry, 15
 - NXgetpath, 15
 - NXinitattdir, 15
 - NXinitgroupdir, 16
 - NXopengrouppath, 16
 - NXopenpath, 16
 - NXopensourcegroup, 16
- General Initialisation and shutdown, 5
 - NXclose, 6
 - NXopen, 6
 - NXreopen, 7
- Linking, 19
 - NXgetdataID, 20
 - NXgetgroupID, 20
 - NXmakelink, 20
 - NXmakenamedlink, 20
 - NXsameID, 21
- Memory allocation, 21
 - NXfree, 21
 - NXmalloc, 22
- Meta data routines, 17
 - NXgetattrinfo, 17
 - NXgetgroupinfo, 17
 - NXgetinfo, 18
 - NXgetrawinfo, 18
 - NXgetversion, 19
 - NXinquirefile, 19
- NXclose
 - General Initialisation and shutdown, 6
- NXclosedata
 - Reading and Writing Data, 9
- NXclosegroup
 - Reading and Writing Groups, 7
- NXcompakedata
 - Reading and Writing Data, 9
- NXcompress
 - Reading and Writing Data, 10
- NXflush
 - Reading and Writing Data, 10
- NXfree
 - Memory allocation, 21
- NXgetattr
 - Reading and Writing Data, 10
- NXgetattrinfo
 - Meta data routines, 17
- NXgetdata
 - Reading and Writing Data, 11
- NXgetdataID
 - Linking, 20
- NXgetgroupID
 - Linking, 20
- NXgetgroupinfo
 - Meta data routines, 17
- NXgetinfo
 - Meta data routines, 18
- NXgetnextattr
 - Reading and Writing Data, 11
- NXgetnextentry
 - General File navigation, 15
- NXgetpath
 - General File navigation, 15
- NXgetrawinfo
 - Meta data routines, 18
- NXgetslab
 - Reading and Writing Data, 12
- NXgetversion
 - Meta data routines, 19
- NXinitattdir

- General File navigation, 15
- NXinitgroupdir
 - General File navigation, 16
- NXinquirefile
 - Meta data routines, 19
- NXisexternaldataset
 - External linking, 22
- NXisexternalgroup
 - External linking, 23
- NXlinkexternal
 - External linking, 23
- NXlinkexternaldataset
 - External linking, 24
- NXmakedata
 - Reading and Writing Data, 12
- NXmakegroup
 - Reading and Writing Groups, 7
- NXmakelink
 - Linking, 20
- NXmakenamedlink
 - Linking, 20
- NXmalloc
 - Memory allocation, 22
- NXopen
 - General Initialisation and shutdown, 6
- NXopendata
 - Reading and Writing Data, 12
- NXopengroup
 - Reading and Writing Groups, 8
- NXopengrouppath
 - General File navigation, 16
- NXopenpath
 - General File navigation, 16
- NXopensourcegroup
 - General File navigation, 16
- NXputattr
 - Reading and Writing Data, 13
- NXputdata
 - Reading and Writing Data, 13
- NXputslab
 - Reading and Writing Data, 13
- NXreopen
 - General Initialisation and shutdown, 7
- NXsameID
 - Linking, 21
- NXsetnumberformat
 - Reading and Writing Data, 14
- Reading and Writing Data, 8
 - NXclosedata, 9
 - NXcompmakedata, 9
 - NXcompress, 10
 - NXflush, 10
 - NXgetattr, 10
 - NXgetdata, 11
 - NXgetnextattr, 11
 - NXgetslab, 12
 - NXmakedata, 12
 - NXopendata, 12
 - NXputattr, 13
 - NXputdata, 13
 - NXputslab, 13
 - NXsetnumberformat, 14
- Reading and Writing Groups, 7
 - NXclosegroup, 7
 - NXmakegroup, 7
 - NXopengroup, 8