NeXus–SWIG Interface

Mark Koennecke

Laboratory for Neutron Scattering

Paul Scherrer Institute

CH–5232 Villigen–PSI

Switzerland

Mark.Koennecke@psi.ch

November 2002

1   Introduction

This is a description of the SWIG interface to the NeXus–API. NeXus is a proposal for a common data format for synchrotron and neutron diffraction data. NeXus uses HDF, the Hierachical Data Format, from the National Center for Super Computing Applications (NCSA) as its physical file format. NeXus files are accessed through a NeXus–API which sits between application programs and the HDF–libraries and then the files themselves. For more information on NeXus see:

http://lns00.psi.ch/NeXus

SWIG is the Simplified Wrapper and Interface Generator. This is a software tool which creates the necessary wrapper code needed to access a given ANSI–C or C++ library from a variety of scripting languages including: Tcl, Java, Perl, Phyton, scheme etc. For more information about SWIG see:

http://www.swig.org

This now is the description of the SWIG interface to the NeXus–API. This document is intended for users who are familiar with the NeXus–API. The meaning of the functions is the same as for the ANSI–C NeXus–API, only function signatures may have changed. Please refer to the NeXus–API documentation for further reference.

2   General Remarks

When interfacing an API like NeXus to a scripting language a couple of issues have to be handled:

pointers   The NeXus–API uses pointers extensively. Fortunately SWIG provides a means for encapsulating pointers in a script language.

memory management   Most scripting language have some kind of automatic variable management or garbage collection. Interfaces generated with SWIG however still use the C memory management. This imples that if you call C–routines which allocate memory you should not forget to free the memory again after you are done. Otherwise you may end up with a system out of memory due to memory leakage.

datasets   NeXus works on possibly large datasets which may have different number types. Support for handling large arrays misses in many scripting languages. Therefore this interface system provides its own implementation of datasets.

return values Some functions of the NeXus–API return more then one value through the call by reference mechanism. These functions had to be wrapped such that all necessary return values are passed back into the scripting language.

Necessarily the generated interface will look slightly different in each target scripting language. For instance the constant NXACC_READ is referred to as:

```
$NXACC_READ
```

in Tcl. In scheme this looks like:

```
(nxacc-read)
```

Functions also follow the calling conventions of the target language. An example in Tcl and scheme:

Tcl set fd [nx_open ''nxinter.hdf'' $NXACC_READ]

mzScheme (define fd (nx-open "nxinter.hdf" (nxacc-read)))

The actual mapping generated for a target scripting language by SWIG can be found by studying SWIG's documentation or through browsing the generated wrapper file for your interface. All examples in the reference below are given in Tcl syntax.

SWIG goes a long way to help in the creation of scripting language interfaces. The user still has to master the process of compiling and linking a scripting language extension and loading it into the target interpreter. Examples for this are provided in the SWIG package.

3   The Dataset Interface

The dataset interface brings multidimensional datasets to the scripting language. This is necessary because many scripting languages do not have a good support for multi dimensional arrays of numbers. If a scripting language supports multidimensional arrays, another extension could be written which transfers such data efficiently, either directly from the NeXus file or using the NeXus dataset as an intermediary. Currently no such optimisations are provided.

There usually is a number type associated with a NeXus dataset. In order to save effort and for simplification the NeXus number types were used. The names are self explaining, if more information about the meaning of these types is required, please consult the NeXus documentation. The types provided are (in Tcl syntax):

```
$NX_FLOAT32
$NX_FLOAT64
$NX_INT8
$NX_UINT8
$NX_INT16
$NX_UINT16
$NX_INT32
$NX_UINT32
$NX_CHAR
```

For notational convenience a symbol **nxdsPtr** is now introduced. This symbol stands for a pointer to a NeXus dataset wrapped according to the scripting languages conventions.

The dataset interface consists of the following functions. Tcl syntax is assumed for this description.

nxdsPtr create_nxds rank type dim0 dim1 dim2 dim3 dim4 dim5 dim6 creates a NeXus dataset with the specified rank and type and the dimensions given as dim0 - dim6. If you do not need so many dimensions, leave the surplus ones out, this sytem will replace the values with zeros. The interface can be easily extended to support more then 7 dimensions if required.

nxdsPtr create_text_nxds textdata convenience function which wraps textdata into a NeXus dataset.

drop_nxds nxdsPtr use this function to dispose of datasets which are no longer needed. Do this, otherwise memory leaks occur!

get_nxds_rank nxdsPtr returns the rank of the dataset.

get_nxds_type nxdsPtr returns the data type of the datset as an integer.

get_nxds_dim nxdsPtr which returns the dimension of the dataset in dimension which.

get_nxds_value nxdsPtr dim0 dim1 dim2 dim3 dim4 dim5 dim6 returns the value of the dataset at the index specified by dim0 - dim6. Again, leave out unneccessary indexes.

get_nxds_text nxdsPtr convenience function which returns the content of the dataset as a text string. his works only if the dataset has rank 1 and is of type NX_CHAR, NX_INT8 or NX_UINT8.

put_nxds_value nxdsPtr val dim0 dim1 dim2 dim3 dim4 dim5 dim6 sets the value of the dataset at the cell denoted by dim0 -dim6 to the value val. Again, you may omit surplu indexes.

4 The NeXus–API Interface

4.1 Notation

After opening them, NeXus files are referred to through a handle. This handle is denoted through the symbol nxFil in the next sections.

There is another symbol **nxSuccess** which stands for an integer. This is 1 if the function returned with success and 0 in case of a failure. If a function returns a pointer, failure is indicated through the NULL pointer. The encoding of the NULL pointer varies between scripting languages.

4.2 Error Handling

Most NeXus-API functions return 1 on success and 0 in case of an error. The exception are those functions which return a pointer. These return a NULL pointer in case of an error. In each case more information about the problem can be obtained by calling:
nx_getlasterror This call returns a string describing the last NeXus error found.

4.3   File Creation accessCode Constants

The meanings of the constants are as described in the NeXus–API documentation.

```
$NXACC_READ
$NXACC_RDWR
$NXACC_CREATE
$NXACC_CREATE4
$NXACC_CREATE5
```

4.4   Opening and Closing of Files

nxFil nx_open filename accessCode opens the NeXus file filename. The accessCodes must be one of the constants given above. Returns a new handle in the case of a success, NULL in case of failure.

nxFil nx_flush nxFil flushes a NeXus file.

nx_close nxFil closes a NeXus file, The handle nxFil is useless after this. his call is necessary, especially when writing files.

4.5   Group Operations

nxSuccess nx_makegroup nxFil name nxclass

nxSuccess nx_opengroup nxFil name nxclass

nxSuccess nx_closegroup nxFil

nxMulti nx_getnextentry nxFil separatorChar performs group directory searches. nxMulti is a string containing name and NeXus class of the group item separated by the character given as separatorChar. If the search ends, NULL is returned.

nxSuccess nx_initgroupdir nxFil

nxPtr nx_getgroupID nxFil returns a pointer to a structure needed for linking.

4.6   Dataset Handling

nxSuccess nx_makedata nxFil name rank type dimDs makes a new dataset. The dimensions are described through the NeXus dataset dimDs.

nxSuccess nx_compmakedata nxFil name rank type dimDs bufds as above, but for compressed datasets. The HDF–5 buffering size is specified through the NeXus dataset bufDs.

nx_opendata nxFil name

nx_closedata nxFil

nx_putslab nxFil nxdsPtr startDs slabbed data writing. The data comes from the NeXus dataset nxdsPtr. The start point from the NeXus dataset startDS. The size of the dataset to write is the size of nxdsPtr.

nxdsPtr nx_getslab nxFil startDs sizeDs reads a slab. startDS and sizeDs are two NeXus datasets describing the slab to read. The data is returned as a NeXus dataset. Do not forget to drop this dataset once you are done with the data!

nxdsPtr nx_getds nxFil reads dataset name. This convenience function opens the dataset, allocates a NeXus dataset of appropriate type and size for you and closes the SDS again.

nxSuccess nx_putds nxFil name nxdsPtr writes the dataset nxdsPtr to the file as name at the current position in the hierarchy. Same convenience fatures as above.

nxdsPtr nx_getdata nxFil normal NeXus getdata but returns a NeXus dataset.

nxSuccess nx_putdata nxFil nxdsPtr normal NeXus putdata. Data is taken from the NeXus dataset nxdsPtr.

nxdsPtr nx_getinfo nxFil returns the current datasets type, rank and dimensions in a one dimensional NeXus dataset.

ptr nx_getdataID nxFil retrieves link pointer for linking.

4.7   Attributes

nxText nx_getnextattr nxFil separatorCharreads the next entry of attribute directory. nxText then contains then name, length and type of the attribute formatted as a string and separated by the character separatorChar.

nxSuccess nx_putattr nxFil name nxdsPtr writes an attribute name from the NeXus dataset nxdsPtr.

nxdsPtr nx_getattr nxFil name type length reads an attribute into a dataset. The data type and the length of the attribute have to specified.

4.8   Making Links

bf nxSuccess nx_makelink nxFil nxLink makes a link. nxLink is one of the pointers returned from the nx_getgroupID or nx_getdataID functions.

5   Example

As an example for the usage of the API see the API test program documented below:

```
#------------------------------------------------------------------------
# Test program for the nxinter interface. Also example usage.
#
# copyright: GPL
#
# Mark Koennecke, October 2002
#------------------------------------------------------------------------

#load ./nxinter.so

#------------- testing dataset interface
set ds [create_nxds 2 $NX_FLOAT32 3 3]

put_nxds_value $ds 1 0 0
put_nxds_value $ds 1 1 1
put_nxds_value $ds 1 2 2

puts stdout "Testing dataset interface "
puts stdout [format "rank = %d" [get_nxds_rank $ds]]
```

```tcl
puts stdout [format "type = %d" [get_nxds_type $ds]]
puts stdout [format "dim1 = %d" [get_nxds_dim $ds 1]]

proc printDS {ds} {
for {set i 0} {$i < 3} {incr i} {
    puts stdout " "
    for {set j 0} {$j < 3} {incr j} {
puts -nonewline stdout [format " %f" [get_nxds_value $ds $i $j]]
    }
}
puts stdout "      "
}

printDS $ds
puts stdout "Hmmmmmmmmhhh....... seems OK"

#-------------- prepare a dimension dataset
set dimds [create_nxds 1 $NX_INT32 2]
put_nxds_value $dimds 3 0
put_nxds_value $dimds 3 1

#--------- prepare slabbing slabber dimensions
set start [create_nxds 1 $NX_INT32 2]
put_nxds_value $start 0 0
put_nxds_value $start 0 1


#--------------------------------- write tests
puts stdout "Testing writing ..........."

set fd [nx_open "nxinter.hdf" $NXACC_CREATE5]
puts stdout [format "Opening file worked: %s" $fd]

#---------- write an attribute.....
set tds [create_text_nxds "Rosa Waschmaschinen sind hip"]
puts stdout [format "Writing SuperDuper = %s" [get_nxds_text $tds]]
puts stdout [format "Writing attribute results in: %d " \
[nx_putattr $fd "SuperDuper" $tds]]
drop_nxds $tds

#----------- making groups....
set status [nx_makegroup $fd fish NXentry]
if {$status == 1} {
    puts stdout "Creating vGroup worked"
}

set status [nx_opengroup $fd fish NXentry]
```

```
if {$status == 1} {
    puts stdout "Opening vGroup worked"
}

set lnk [nx_getgroupID $fd]
puts stdout [format "groupID determined to: %s" $lnk]

#------------ writing tata
puts stdout "Test Writing data....."
puts stdout [nx_makedata $fd "fish" 2 $NX_FLOAT32 $dimds]
puts stdout [nx_opendata $fd "fish"]
puts stdout [nx_putdata $fd $ds]
set lnk [nx_getdataID $fd]
puts stdout $lnk
puts stdout [nx_closedata $fd]

#--------------- testing slabbed tata writing
puts stdout "Testing  writing in slabs"
put_nxds_value $dimds 6 0
puts stdout [format "Dimensions for slab test: %f, %f"  \
[get_nxds_value $dimds 0] [get_nxds_value $dimds 1]]

puts stdout [nx_makedata $fd "fish2" 2 $NX_FLOAT32 $dimds]
puts stdout [nx_opendata $fd "fish2"]
puts stdout [nx_putslab  $fd $ds $start]
put_nxds_value $start 3 0
puts stdout [nx_putslab  $fd $ds $start]
puts stdout [nx_closedata $fd]
puts stdout "Finished Writing Slabs........."

puts stdout [format "Linking = %d" [nx_makelink $fd $lnk]]

set status [nx_closegroup $fd]
if {$status == 1} {
    puts stdout "Closing vGroup worked"
}


nx_close $fd
puts stdout "Closed file"
#--------------- finished writing tests


#--------------- trying to read
puts stdout "Testing Reading files"

set fd [nx_open "nxinter.hdf" $NXACC_READ]
```

```tcl
puts stdout "Opening file for reading worked"

set run 1

#---------------- printing group content
puts stdout "Group directory listing"
while {$run == 1} {
    set entry [nx_getnextentry $fd / ]
    if { [string length $entry] < 2 } {
set run 0
    } else {
puts stdout $entry
    }
}
#--------- printing attributes
puts stdout "Attributes"
set run 1
while {$run == 1} {
    set entry [nx_getnextattr $fd / ]
    puts stdout $entry
    if { [string length $entry] < 2 } {
set run 0
    }
}

set rds [nx_getattr $fd "SuperDuper" $NX_CHAR 30]
puts stdout $rds
if {[string compare $rds NULL] != 0} {
    puts stdout [format "SuperDuper = %s" [get_nxds_text $rds]]
}
drop_nxds $rds

#---------------- reading tata
puts stdout [nx_opengroup $fd fish NXentry]
puts stdout [nx_opendata $fd "fish"]
set rds [nx_getdata $fd]
puts stdout $rds

puts stdout "Read data should be a 3x3 unity matrix"
printDS $rds

puts stdout [nx_closedata $fd]

#----------- reading slabbed data
puts stdout "Testing slabbed reading......"
puts stdout [nx_opendata $fd "fish2"]
set ids [nx_getinfo $fd]
```

8

```
puts stdout [format "fish2: type %f, rank %f, d1 %f, d2 %f" \
[get_nxds_value $ids 0] [get_nxds_value $ids 1] \
[get_nxds_value $ids 2] [get_nxds_value $ids 3]]
drop_nxds $ids

put_nxds_value $dimds 3 0
set r1 [nx_getslab $fd $start $dimds]
printDS $r1
put_nxds_value $start 0 0
set r2 [nx_getslab $fd $start $dimds]
printDS $r2
puts stdout [nx_closedata $fd]

puts stdout [nx_closegroup $fd]
puts stdout [nx_close $fd]

#--------------- dropping datasets: do not forget!!!
drop_nxds $ds
drop_nxds $rds
drop_nxds $dimds
```