The NetCDF C++ Interface Guide

Class Documentation
Version 4.1.3
30 June 2011

Russ Rew
Unidata Program Center

Copyright © 2005-2009 University Corporation for Atmospheric Research

Permission is granted to make and distribute verbatim copies of this manual provided that the copyright notice and these paragraphs are preserved on all copies. The software and any accompanying written materials are provided "as is" without warranty of any kind. UCAR expressly disclaims all warranties of any kind, either expressed or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose.

The Unidata Program Center is managed by the University Corporation for Atmospheric Research and sponsored by the National Science Foundation. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Mention of any commercial company or product in this document does not constitute an endorsement by the Unidata Program Center. Unidata does not authorize any use of information from this publication for advertising or publicity purposes.

Table of Contents

Introduction **1**
Class Hierarchy 1
Auxiliary Types and Constants 2

NetCDF Classes **3**
Class NcFile 3
Public Member Functions 3
Class NcDim 5
Public Member Functions 6
Class NcTypedComponent 6
Public Member Functions 6
Class NcVar 7
Public Member Functions 7
Class NcAtt 12
Public Member Functions 12

Auxiliary Classes **15**
Class NcValues 15
Public Member Functions 15
Class NcError 16
Public Member Functions 16

Index **17**

Introduction

The main requirements for the design of the C++ interface are:

- to provide the functionality of the C interface;
- to provide type safety by eliminating all use of `void*` pointers; and
- to provide an interface that is simpler to use than the C interface.

Some of the features of the C++ interface are:

- No IDs needed for netCDF's variables, dimensions, or attributes.
- No explicit open or close calls needed for netCDF files; a constructor opens and a destructor closes a file.
- No need to specify types for creating attributes; they will have the type of the value provided.
- No use of `void*`: values are type-checked.
- Less indirection is needed for dimensions and dimension sizes than with the C interface. A variable's dimensions can be provided as arguments when defining a variable.
- Code for data types is isolated to make the addition of new types easier.
- No explicit `ncredef` or `ncendef` calls are needed for switching between define and data modes. Whenever a mode switch is required, it happens implicitly.

The header file '`netcdfcpp.h`' must be included in source code files using this interface.

This release provides some of the functionality of netCDF version 4, but not for the enhanced data model introduced with netCDF-4.

This manual assumes familiarity with the netCDF User's Guide, where the concepts of netCDF dimensions, variables, and attributes are discussed.

Class Hierarchy

The class for netCDF file objects is `NcFile`.

The components of a netCDF file are dimensions, variables, and attributes. There is a class for each of these kinds of objects; `NcDim`, `NcVar`, and `NcAtt`. Variables and attributes share some common characteristics that are factored out in the abstract base class `NcTypedComponent`.

An auxiliary class, `NcValues`, provides a type for arrays of values that are read from or written to netCDF files. Another auxiliary class, `NcError`, provides facilities for handling errors.

```
NcFile                      netCDF file

NcDim                       dimension

NcTypedComponent            abstract base class
        NcVar                 variable
        NcAtt                 attribute

NcValues                    abstract base class for array
        NcValues_ncbyte       array of bytes
        NcValues_char         array of characters
        NcValues_short        array of shorts
        NcValues_int          array of ints
        NcValues_long         array of longs
        NcValues_float        array of floats
        NcValues_double       array of doubles

NcError                     for error handling
```

Auxiliary Types and Constants

The netCDF classes use several auxiliary types for arguments and return types from member functions: `NcToken`, `NcType`, `NcBool`, and `ncbyte`.

`NcToken` Used for names for netCDF objects, in particular variable names, dimension names, and attribute names. Currently this is just a typedef for `const char*`.

`NcType` Used for specifying netCDF external value types. Currently this is an enumerated type with the following legitimate values: `ncByte`, `ncChar`, `ncShort`, `ncInt`, `ncLong` (deprecated), `ncFloat`, and `ncDouble`.

`NcBool` Used for the return type of some member functions. If the member function fails, 0 is returned, otherwise some non-zero value. Currently this is just a typedef for `unsigned int`. It will be changed to `bool` when all C++ compilers support the new `bool` type.

`ncbyte` Used to declare values of type `ncByte`, for 8-bit integer data. (This is currently a typedef for `unsigned char`, but it may be changed to a typedef for `signed char`, so don't depend on the underlying representation.)

NetCDF Classes

Class NcFile

`NcFile` is the class for netCDF files, providing methods for netCDF file operations.

Some member functions return pointers to dimensions (`NcDim`) or variables (`NcVar`). These objects are owned by the `NcFile` they are associated with, and will be deleted automatically by the `NcFile` destructor (or by the `close` member function, if this is called earlier than the destructor), so users should not delete these. Member functions that return pointers to attributes (`NcAtt`) pass ownership to the calling function; users should delete attributes when they are finished with them.

Member functions that return `NcBool` yield `TRUE` on success and `FALSE` on failure. Member functions that return a pointer value return a `NULL` pointer on failure.

This class interface hides the distinction in the C and Fortran interfaces between *define mode* (when dimensions, variables, or attributes are being defined or renamed), and *data mode* (when data values are being accessed), by automatically switching between the modes when necessary. Be aware that switching from accessing data to adding or renaming dimensions, variables and attributes can be expensive, since it may entail a copy of the data.

Public Member Functions

`NcFile( const char * path, FileMode = ReadOnly, size_t *chunksizeptr = NULL, size_t initialsize = 0, FileFormat = Classic)`

> The constructor creates a new netCDF file or opens an existing netCDF file. The path argument may be an OPenDAP DAP URL if DAP support is enabled.
>
> The `FileMode` argument can be any of `ReadOnly` (the default) to open an existing file for reading, `Write` to open an existing file for reading or writing, `Replace` to create a new empty file even if the named file already exists, or `New` to create a new file only if the named file does not already exist.
>
> The optional `FileFormat` argument can be any of `Classic` (the default), `Offset64Bits`, `Netcdf4`, or `Netcdf4Classic`.
>
> The optional `chunksizeptr` and `initialsize` tuning parameters are as described in the corresponding `nc__create()` function in the C interface.
>
> The constructor will not fail, but in the case of a bad path name, improper permissions, or if the file already exists and you have specified `FileMode` as `New`, no netCDF file will be created or opened. If the constructor fails to create or open a netCDF file, a subsequent call to the `is_valid()` member function will return `False`.

`~NcFile( void )`

> Destructor. The file is closed and all resources associated with it are released, including the associated `NcVar` and `NcDim` objects. If you wish to close the file earlier, you may explicitly call the `close` member function; a subsequent destructor call will work properly.

`NcBool close( void )`
: Close netCDF file earlier than it would be closed by the `NcFile` destructor.

`NcBool is_valid( void ) const`
: Returns `TRUE` if valid netCDF file, `FALSE` otherwise (e.g. if constructor could not open file).

`int num_dims( void ) const`
: Returns the number of dimensions in the netCDF file.

`int num_vars( void ) const`
: Returns the number of variables in the netCDF file.

`int num_atts( void ) const`
: Returns the number of global attributes in the netCDF file.

`NcDim* get_dim(NcToken name) const`
: Get a dimension by name.

`NcVar* get_var(NcToken name) const`
: Get a variable by name.

`NcAtt* get_att(NcToken name) const`
: Get a global attribute by name.

`NcDim* get_dim(int n) const`
: Get the *n*th dimension (beginning with the 0th).

`NcVar* get_var(int n) const`
: Get the *n*th variable (beginning with the 0th).

`NcAtt* get_att(int n) const`
: Get the *n*th global attribute (beginning with the 0th).

`NcDim* rec_dim( void ) const`
: Get the unlimited dimension, if any.

The following `add_` member functions put the file in *define mode*, so could be expensive. To avoid copying of data, invoke these before writing data to variables.

`NcDim* add_dim(NcToken dimname)`
: Add an unlimited dimension named `dimname` to the netCDF file.

`NcDim* add_dim(NcToken dimname, long dimsize)`
: Add a dimension named `dimname` of size `dimsize`.

`NcVar* add_var(NcToken varname, NcType type, const NcDim*, ...)`
: Add a variable named `varname` of the specified type (`ncByte`, `ncChar`, `ncShort`, `ncInt`, `ncFloat`, `ncDouble`) to the open netCDF file. The variable is defined with a shape that depends on how many dimension arguments are provided. A scalar variable would have 0 dimensions, a vector would have 1 dimension, and so on. Supply as many dimensions as needed, up to 5. If more than 5 dimensions are required, use the n-dimensional version of this member function instead.

`NcVar* add_var(NcToken varname, NcType type, int ndims, const NcDim** dims)`
: Add a variable named `varname` of `ndims` dimensions and of the specified type. This method must be used when dealing with variables of more than 5 dimensions.

`NcBool add_att(NcToken name, ncbyte val)`
`NcBool add_att(NcToken name, char val)`
`NcBool add_att(NcToken name, short val)`
`NcBool add_att(NcToken name, int val)`
`NcBool add_att(NcToken name, float val)`
`NcBool add_att(NcToken name, double val)`
: Add global scalar attributes of the specified name and with the supplied value.

`NcBool add_att(NcToken name, const char* val)`
: Add global string-valued attribute with the specified name and C string value (terminated with a `\0` character).

`NcBool add_att(NcToken name, int n, const ncbyte* val)`
`NcBool add_att(NcToken name, int n, const char* val)`
`NcBool add_att(NcToken name, int n, const short* val)`
`NcBool add_att(NcToken name, int n, const int* val)`
`NcBool add_att(NcToken name, int n, const float* val)`
`NcBool add_att(NcToken name, int n, const double* val)`
: Add global vector attributes with the specified name, length, and values.

`NcBool set_fill(FillMode mode = Fill)`
: Sets fill-mode to either `NcFile::Fill` or `NcFile::NoFill`. Default is `Fill`, in which case unwritten values are pre-written with appropriate type-specific or variable-specific fill values.

`enum NcFile::FillMode get_fill( void ) const`
: Returns fill mode of the file, either `NcFile::Fill` or `NcFile::NoFill`.

`enum NcFile::FileFormat get_format( void ) const`
: Returns format version of the file, either `NcFile::Classic`, `NcFile:Offset64Bits`, `NcFile:Netcdf4`, or `NcFile::Netcdf4Classic`.

`NcBool sync( void )`
: Synchronizes file to disk. This flushes buffers so that readers of the file will see recent changes.

`NcBool abort( void )`
: Either just closes file (if recently it has been in data mode as the result of accessing data), or backs out of the most recent sequence of changes to the file schema (dimensions, variables, and attributes).

Class NcDim

A netCDF dimension has a name and a size. Dimensions are only created and destroyed by NcFile member functions, because they cannot exist independently of an open netCDF file. Hence there are no public constructors or destructors.

Public Member Functions

`NcToken name( void ) const`
: Returns the name of the dimension if it exists, 0 otherwise.

`long size( void ) const`
: Returns the dimension size.

`NcBool is_valid( void ) const`
: Returns `TRUE` if file and dimension are both valid, `FALSE` otherwise.

`NcBool is_unlimited( void ) const`
: Returns `TRUE` if the dimension is the unlimited dimension, `FALSE` if either not a valid netCDF file, or if the dimension is not the unlimited dimension.

`NcBool rename( NcToken newname )`
: Renames the dimension to `newname`.

`NcBool sync( void )`
: If the dimension may have been renamed, make sure its name is updated.

Class NcTypedComponent

`NcTypedComponent` is an abstract base class for `NcVar` and `NcAtt` that captures the similarities between netCDF variables and attributes. We list here the member functions that variables and attributes inherit from `NcTypedComponent`, but these member functions are also documented under the `NcVar` and `NcAtt` classes for convenience.

Public Member Functions

`NcToken name( void ) const`
: Returns the name of the variable or attribute.

`NcType type( void ) const`
: Returns the type of the variable or attribute. The type will be one of `ncByte`, `ncChar`, `ncShort`, `ncInt`, `ncFloat`, or `ncDouble`.

`NcBool is_valid( void ) const`
: Returns `TRUE` if the component is valid, `FALSE` otherwise.

`long num_vals( void ) const`
: Returns the number of values for an attribute or variable. For an attribute, this is just 1 for a scalar attribute, the number of values for a vector-valued attribute, and the number of characters for a string-valued attribute. For a variable, this is the product of the dimension sizes for all the variable's dimensions.

`NcBool rename( NcToken newname )`
: Renames the variable or attribute.

`NcValues* values( void ) const`
: Returns a pointer to the block of all values for the variable or attribute. The caller is responsible for deleting this block of values when no longer needed, as well as the pointer returned by the `values` method. Note that this is not a good way to read selected values of a variable; use the `get` member function instead, to get single values or selected cross-sections of values.

`ncbyte as_ncbyte( int n ) const`
`char as_char( int n ) const`
`short as_short( int n ) const`
`int as_int( int n ) const`
`nclong as_nclong( int n ) const // deprecated`
`long as_long( int n ) const`
`float as_float( int n ) const`
`double as_double( int n ) const`
`char* as_string( int n ) const`

Get the n-th value of the attribute or variable. These member functions provide conversions from the value type of the variable or attribute to the specified type. If the value is out-of-range, the fill-value of the appropriate type is returned.

Class NcVar

`NcVar` is derived from NcTypedComponent, and represents a netCDF variable. A netCDF variable has a name, a type, a shape, zero or more attributes, and a block of values associated with it. Because variables are only associated with open netCDF files, there are no public constructors for this class. Use member functions of `NcFile` to get variables or add new variables.

Public Member Functions

`NcToken name( void ) const`

Returns the name of the variable.

`NcType type( void ) const`

Returns the type of the variable. The type will be one of `ncByte`, `ncChar`, `ncShort`, `ncInt`, `ncFloat`, or `ncDouble`.

`int num_dims( void ) const`

Returns number of dimensions for this variable.

`NcDim* get_dim( int n ) const`

Returns a pointer to the n-th dimension (starting at 0). Returns a NULL-pointer if an invalid dimension is requested.

`long* edges( void ) const`

Returns the shape of the variable, in the form of a vector containing the sizes of the dimensions of the variable. The caller is responsible for deleting the returned edge vector when no longer needed.

`int num_atts( void ) const`

Returns the number of attributes attached to the variable.

`NcAtt* get_att( NcToken attname ) const`
`NcAtt* get_att( int n ) const`

The first member function returns a variable attribute by name. The second returns the n-th (starting at 0) attribute of the variable. In either case, if no such attribute has been attached to the variable, zero is returned. Attributes returned in this way belong to the caller, and hence should eventually be deleted by the caller to avoid memory leaks.

`NcBool is_valid( void ) const`
: Returns `TRUE` if the variable is valid, `FALSE` otherwise.

`long num_vals( void ) const`
: Returns the number of values for a variable. This is just 1 for a scalar variable, or the product of the dimension sizes for all the variable's dimensions.

`NcValues* values( void ) const`
: Returns a pointer to the block of all values for the variable. The caller is responsible for deleting this block of values when no longer needed. Note that this is not a good way to read selected values of a variable; use the `get` member function instead, to get single values or selected cross-sections of values.

```
NcBool put(const ncbyte* vals, long c0, long c1, long c2, long c3, long c4)
NcBool put(const char* vals, long c0, long c1, long c2, long c3, long c4)
NcBool put(const short* vals, long c0, long c1, long c2, long c3, long c4)
NcBool put(const int* vals, long c0, long c1, long c2, long c3, long c4)
NcBool put(const long* vals, long c0, long c1, long c2, long c3, long c4)
NcBool put(const float* vals, long c0, long c1, long c2, long c3, long c4)
NcBool put(const double* vals, long c0, long c1, long c2, long c3, long c4)
```
: Write scalar or 1 to 5-dimensional arrays by providing enough arguments. The `vals` argument points to a contiguous block of values in memory to be written. This means if you allocate each row of an array with a "new" call for example, you must not write more than one row at a time, because the rows may not be contiguous in memory. Other arguments are edge lengths, and their number must not exceed variable's dimensionality. Start corner is `[0,0,..., 0]` by default, but may be reset using the `set_cur()` member function for this variable. `FALSE` is returned if type of values does not match type for variable. For more than 5 dimensions, use the overloaded n-dimensional form of the `put` member function.

```
NcBool put(const ncbyte* vals, const long* counts)
NcBool put(const char* vals, const long* counts)
NcBool put(const short* vals, const long* counts)
NcBool put(const int* vals, const long* counts)
NcBool put(const long* vals, const long* counts)
NcBool put(const float* vals, const long* counts)
NcBool put(const double* vals, const long* counts)
```
: Write n-dimensional arrays, starting at `[0, 0, ..., 0]` by default, may be reset with `set_cur()`. `FALSE` is returned if type of values does not match type for variable. The `vals` argument points to a contiguous block of values in memory to be written. This means if you allocate each row of an array with a "new" call for example, you must not write more than one row at a time, because the rows may not be contiguous in memory.

```
NcBool get(ncbyte* vals, long c0, long c1, long c2, long c3, long c4) const
NcBool get(char* vals, long c0, long c1, long c2, long c3, long c4) const
NcBool get(short* vals, long c0, long c1, long c2, long c3, long c4) const
NcBool get(int* vals, long c0, long c1, long c2, long c3, long c4) const
NcBool get(long* vals, long c0, long c1, long c2, long c3, long c4) const
NcBool get(float* vals, long c0, long c1, long c2, long c3, long c4) const
NcBool get(double* vals, long c0, long c1, long c2, long c3, long c4) const
```

Get scalar or 1 to 5 dimensional arrays by providing enough arguments. The `vals` argument points to a contiguous block of values in memory into which values will be read. This means if you allocate each row of an array with a "new" call for example, you must not read more than one row at a time, because the rows may not be contiguous in memory. Other arguments are edge lengths, and their number must not exceed variable's dimensionality. Start corner is `[0,0,..., 0]` by default, but may be reset using the `set_cur()` member function. `FALSE` is returned if type of values does not match type for variable.

```
NcBool get(ncbyte* vals, const long* counts) const
NcBool get(char* vals, const long* counts) const
NcBool get(short* vals, const long* counts) const
NcBool get(int* vals, const long* counts) const
NcBool get(long* vals, const long* counts) const
NcBool get(float* vals, const long* counts) const
NcBool get(double* vals, const long* counts) const
```

Get n-dimensional arrays, starting at `[0, 0, ..., 0]` by default, may be reset with `set_cur()` member function. `FALSE` is returned if type of values does not match type for variable. Get scalar or 1 to 5 dimensional arrays by providing enough arguments. The `vals` argument points to a contiguous block of values in memory into which values will be read. This means if you allocate each row of an array with a "new" call for example, you must not read more than one row at a time, because the rows may not be contiguous in memory.

```
NcBool set_cur(long c0=-1, long c1=-1, long c2=-1, long c3=-1, long c4=-1)
NcBool set_cur(long* cur)
```

Resets the starting corner to the values supplied. The first form works for a variable of dimensionality from scalar to 5 dimensions. For more than five dimensions, use the second form, in which the number of longs supplied must match the rank of the variable. The method returns FALSE if any argument is greater than the size of the corresponding dimension.

```
NcBool add_att( NcToken, char )
NcBool add_att( NcToken, ncbyte )
NcBool add_att( NcToken, short )
NcBool add_att( NcToken, int )
NcBool add_att( NcToken, long )
NcBool add_att( NcToken, float )
NcBool add_att( NcToken, double )
NcBool add_att( NcToken, const char* )
NcBool add_att( NcToken, int, const char* )
NcBool add_att( NcToken, int, const ncbyte* )
NcBool add_att( NcToken, int, const short* )
NcBool add_att( NcToken, int, const int* )
NcBool add_att( NcToken, int, const long* )
NcBool add_att( NcToken, int, const float* )
NcBool add_att( NcToken, int, const double* )
```

Add scalar or vector attribute of any type to a variable, given the name, number of values, and the vector of values. These put file in define mode, so could be expensive. To avoid the expense of copying data, add attributes to variables before writing data.

```
NcBool rename( NcToken newname )
```

Renames the variable. If variable is renamed to a longer name, this puts file in define mode, so could be expensive.

```
ncbyte as_ncbyte( int n ) const
char as_char( int n ) const
short as_short( int n ) const
int as_int( int n ) const
nclong as_nclong( int n ) const // deprecated
long as_long( int n ) const
float as_float( int n ) const
double as_double( int n ) const
char* as_string( int n ) const
```

Get the n-th value of the variable, ignoring its shape. These member functions provide conversions from the value type of the variable to the specified type. If the requested value is out-of-range, the fill-value of the appropriate type is returned.

```
int id( void ) const
```

Return the variable number. This is not needed in the C++ interface, but might be needed in calling a C-function that requires that a variable be identified by number instead of name.

```
NcBool sync( void )
```

If the variable may have been renamed, make sure its name is updated.

```
~NcVar( void )
```

Destructor.

The following member functions are intended for record variables. They will also work for non-record variables, if the first dimension is interpreted as the record dimension.

```
long rec_size( void )
long rec_size( NcDim* )
```
Return the number of values per record or the number of values per dimension slice for the specified dimension.

```
NcValues* get_rec( void )
NcValues* get_rec( long n )
```
Get the data for this variable for the current record or for the *n*th record.

```
NcValues* get_rec( NcDim* )
NcValues* get_rec( NcDim*, long n )
```
Get the data for this variable for the current dimension slice or for the *n*th dimension slice.

```
NcBool put_rec( const ncbyte* vals )
NcBool put_rec( const char* vals )
NcBool put_rec( const short* vals )
NcBool put_rec( const int* vals )
NcBool put_rec( const long* vals )
NcBool put_rec( const float* vals )
NcBool put_rec( const double* vals )
```
Put a record's worth of data for this variable in the current record.

```
NcBool put_rec( NcDim*, const ncbyte* vals )
NcBool put_rec( NcDim*, const char* vals )
NcBool put_rec( NcDim*, const short* vals )
NcBool put_rec( NcDim*, const int* vals )
NcBool put_rec( NcDim*, const long* vals )
NcBool put_rec( NcDim*, const float* vals )
NcBool put_rec( NcDim*, const double* vals )
```
Put a dimension slice worth of data for this variable in the current dimension slice.

```
NcBool put_rec( const ncbyte* vals, long rec )
NcBool put_rec( const char* vals, long rec )
NcBool put_rec( const short* vals, long rec )
NcBool put_rec( const int* vals, long rec )
NcBool put_rec( const long* vals, long rec )
NcBool put_rec( const float* vals, long rec )
NcBool put_rec( const double* vals, long rec )
```
Put a record's worth of data for this variable in the specified record.

```
NcBool put_rec( NcDim*, const ncbyte* vals, long slice )
NcBool put_rec( NcDim*, const char* vals, long slice )
NcBool put_rec( NcDim*, const short* vals, long slice )
NcBool put_rec( NcDim*, const int* vals, long slice )
NcBool put_rec( NcDim*, const long* vals, long slice )
NcBool put_rec( NcDim*, const float* vals, long slice )
NcBool put_rec( NcDim*, const double* vals, long slice )
```
Put a dimension slice worth of data for this variable in the specified dimension slice.

```
long get_index( const ncbyte* vals )
long get_index( const char* vals )
long get_index( const short* vals )
long get_index( const int* vals )
long get_index( const long* vals )
long get_index( const float* vals )
long get_index( const double* vals )
```
Get first record index for this variable corresponding to the specified key value(s).

```
long get_index( NcDim*, const ncbyte* vals )
long get_index( NcDim*, const char* vals )
long get_index( NcDim*, const short* vals )
long get_index( NcDim*, const int* vals )
long get_index( NcDim*, const long* vals )
long get_index( NcDim*, const float* vals )
long get_index( NcDim*, const double* vals )
```
Get first index of specified dimension for this variable corresponding to the specified key value(s).

```
void set_rec ( long rec )
```
Set the current record for this variable.

```
void set_rec ( NcDim*, long rec )
```
Set the current dimension slice for the specified dimension for this variable.

Class NcAtt

`NcAtt` is derived from `NcTypedComponent`, and represents a netCDF attribute. A netCDF attribute has a name and a type, and may be either a scalar attribute or a vector attribute. Scalar attributes have one value and vector attributes have multiple values. In addition, each attribute is attached to a specific netCDF variable or is global to an entire netCDF file. Because attributes are only associated with open netCDF files, there are no public constructors for this class. Use member functions of `NcFile` and `NcVar` to get netCDF attributes or add new attributes. Most of the useful member functions for `NcAtt` are inherited from class `NcTypedComponent`.

Public Member Functions

```
NcToken name( void ) const
```
Returns the name of the attribute.

```
NcType type( void ) const
```
Returns the type of the attribute. The type will be one of `ncByte`, `ncChar`, `ncShort`, `ncInt`, `ncFloat`, or `ncDouble`.

```
NcBool is_valid( void ) const
```
Returns `TRUE` if the attribute is valid, `FALSE` otherwise.

`long num_vals( void ) const`
: Returns the number of values for an attribute. This is just 1 for a scalar attribute, the number of values for a vector-valued attribute, and the number of characters for a string-valued attribute.

`NcBool rename( NcToken newname )`
: Renames the attribute.

`NcValues* values( void ) const`
: Returns a pointer to the block of all values for the attribute. The caller is responsible for deleting this block of values when no longer needed.

```
ncbyte as_ncbyte( int n ) const
char as_char( int n ) const
short as_short( int n ) const
int as_int( int n ) const
nclong as_nclong( int n ) const // deprecated
long as_long( int n ) const
float as_float( int n ) const
double as_double( int n ) const
char* as_string( int n ) const
```
: Get the n-th value of the attribute. These member functions provide conversions from the value type of the attribute to the specified type. If the value is out-of-range, the fill-value of the appropriate type is returned.

`NcBool remove( void )`
: Deletes the attribute from the file and detaches it from the variable. Does not call the destructor. Subsequent calls to `is_valid()` will return `FALSE`.

`~NcAtt( void )`
: Destructor.

Auxiliary Classes

Auxiliary classes include the abstract base class `NcValues`, its type-specific derived subclasses, and the error-handling class `NcError`.

Class NcValues

Class `NcValues` is an abstract base class for a block of typed values. The derived classes are `NcValues_ncbyte`, `NcValues_char`, `NcValues_short`, `NcValues_int`, `NcValues_nclong` (deprecated), and `NcValues_long`, `NcValues_float`, `NcValues_double`. These classes are used as the return type of the `NcTypedComponent::values()` member function, for typed-value arrays associated with variables and attributes.

Public Member Functions

`NcValues( void )`
: Default constructor.

`NcValues(NcType, long)`
: Constructor for a value block of the specified type and length.

`~NcValues( void )`
: Destructor.

`long num( void )`
: Returns the number of values in the value block.

`ostream& print(ostream&) const`
: Used to print the comma-delimited sequence of values of the value block.

`void* base( void ) const`
: Returns a bland pointer to the beginning of the value block.

`int bytes_for_one( void ) const`
: Returns the number of bytes required for one value.

```
ncbyte as_ncbyte( int n ) const
char as_char( int n ) const
short as_short( int n ) const
int as_int( int n ) const
nclong as_nclong( int n ) const // deprecated
long as_long( int n ) const
float as_float( int n ) const
double as_double( int n ) const
char* as_string( int n ) const
```
: Provide conversions for the nth value from the value type to a desired basic type. If the value is out of range, the default "fill-value" for the appropriate type is returned.

Class NcError

This class provides control for netCDF error handling. Declaring an `NcError` object temporarily changes the error-handling behavior for all netCDF classes until the `NcError` object is destroyed (typically by going out of scope), at which time the previous error-handling behavior is restored.

Public Member Functions

`NcError( Behavior b = verbose_fatal )`

The constructor saves the previous error state for restoration when the destructor is invoked, and sets a new specified state. Valid error states are `NcError::silent_nonfatal`, `NcError::verbose_nonfatal`, `NcError::silent_fatal`, or `NcError::verbose_fatal`, to control whether error messages are output from the underlying library and whether such messages are fatal or nonfatal.

`~NcError( void )`

Destructor, restores previous error state.

`int get_err( void )`

Returns most recent error, as enumerated in '`netcdf.h`'.

Index

~

~NcAtt 13
~NcError 16
~NcFile 3
~NcValues 15
~NcVar 10

A

abort 5
add_att 5, 9
add_dim 4
add_var 4
as_char 7, 10, 13, 15
as_double 7, 10, 13, 15
as_float 7, 10, 13, 15
as_int 7, 10, 13, 15
as_long 7, 10, 13, 15
as_ncbyte 6, 10, 13, 15
as_nclong 7, 10, 13, 15
as_short 7, 10, 13, 15
as_string 7, 10, 13, 15
auxiliary types and constants 2

B

base 15
bytes_for_one 15

C

class hierarchy 1
close 3

E

edges 7

G

get 8, 9
get_att 4, 7
get_dim 4, 7
get_err 16
get_fill 5
get_format 5
get_index 11
get_rec 11
get_var 4

I

id 10
is_unlimited 6
is_valid 4, 6, 7, 12

N

name 6, 7, 12
NcAtt 12
NcAtt::~NcAtt 13
NcAtt::remove 13
NcBool 2
ncbyte 2
NcDim 5
NcDim::is_unlimited 6
NcDim::is_valid 6
NcDim::name 6
NcDim::rename 6
NcDim::size 6
NcDim::sync 6
NcError 15, 16
NcError::~NcError 16
NcError::get_err 16
NcFile 3
NcFile::~NcFile 3
NcFile::abort 5
NcFile::add_att 5
NcFile::add_dim 4
NcFile::add_var 4
NcFile::close 3
NcFile::get_att 4
NcFile::get_dim 4
NcFile::get_fill 5
NcFile::get_format 5
NcFile::get_var 4
NcFile::is_valid 4
NcFile::NcFile 3
NcFile::num_atts 4
NcFile::num_dims 4
NcFile::num_vars 4
NcFile::rec_dim 4
NcFile::set_fill 5
NcFile::sync 5
NcToken 2
NcType 2
NcTypedComponent 6
NcTypedComponent::as_char 7, 10, 13
NcTypedComponent::as_double 7, 10, 13
NcTypedComponent::as_float 7, 10, 13
NcTypedComponent::as_int 7, 10, 13
NcTypedComponent::as_long 7, 10, 13
NcTypedComponent::as_ncbyte 6, 10, 13
NcTypedComponent::as_nclong 7, 10, 13
NcTypedComponent::as_short 7, 10, 13
NcTypedComponent::as_string 7, 10, 13
NcTypedComponent::is_valid 6, 7, 12
NcTypedComponent::name 6, 7, 12
NcTypedComponent::num_vals 6, 8, 12

NcTypedComponent::rename 6, 10, 13
NcTypedComponent::type 6, 7, 12
NcTypedComponent::values 6, 8, 13
NcValues 15
NcValues::~NcValues 15
NcValues::as_char 15
NcValues::as_double 15
NcValues::as_float 15
NcValues::as_int 15
NcValues::as_long 15
NcValues::as_ncbyte 15
NcValues::as_nclong 15
NcValues::as_short 15
NcValues::as_string 15
NcValues::base 15
NcValues::bytes_for_one 15
NcValues::NcValues 15
NcValues::num 15
NcValues::print 15
NcValues_char 15
NcValues_double 15
NcValues_float 15
NcValues_int 15
NcValues_long 15
NcValues_ncbyte 15
NcValues_nclong 15
NcValues_short 15
NcVar 7
NcVar::~NcVar 10
NcVar::add_att 9
NcVar::edges 7
NcVar::get 8, 9
NcVar::get_att 7
NcVar::get_dim 7
NcVar::get_index 11
NcVar::get_rec 11
NcVar::id 10
NcVar::num_atts 7
NcVar::num_dims 7
NcVar::put 8
NcVar::put_rec 11
NcVar::rec_size 11
NcVar::set_cur 9
NcVar::set_rec 12
NcVar::sync 10
num 15
num_atts 4, 7
num_dims 4, 7
num_vals 6, 8, 12
num_vars 4

P

print 15
put 8
put_rec 11

R

rec_dim 4
rec_size 11
remove 13
rename 6, 10, 13
requirements of C++ interface 1

S

set_cur 9
set_fill 5
set_rec 12
size 6
sync 5, 6, 10

T

type 6, 7, 12

V

values 6, 8, 13