The NetCDF Fortran 77 Interface Guide

NetCDF Version 4.1.3
June 2011

Russ Rew, Glenn Davis, Steve Emmerson, and Harvey Davies
Unidata Program Center

Table of Contents

1 Use of the NetCDF Library

You can use the netCDF library without knowing about all of the netCDF interface. If you are creating a netCDF dataset, only a handful of routines are required to define the necessary dimensions, variables, and attributes, and to write the data to the netCDF dataset. (Even less are needed if you use the ncgen utility to create the dataset before running a program using netCDF library calls to write data. See Section "ncgen" in *The NetCDF Users Guide*.) Similarly, if you are writing software to access data stored in a particular netCDF object, only a small subset of the netCDF library is required to open the netCDF dataset and access the data. Authors of generic applications that access arbitrary netCDF datasets need to be familiar with more of the netCDF library.

In this chapter we provide templates of common sequences of netCDF calls needed for common uses. For clarity we present only the names of routines; omit declarations and error checking; omit the type-specific suffixes of routine names for variables and attributes; indent statements that are typically invoked multiple times; and use ... to represent arbitrary sequences of other statements. Full parameter lists are described in later chapters.

1.1 Creating a NetCDF Dataset

Here is a typical sequence of netCDF calls used to create a new netCDF dataset:

```
NF_CREATE           ! create netCDF dataset: enter define mode
     ...
   NF_DEF_DIM       ! define dimensions: from name and length
     ...
   NF_DEF_VAR       ! define variables: from name, type, dims
     ...
   NF_PUT_ATT       ! assign attribute values
     ...
NF_ENDDEF           ! end definitions: leave define mode
     ...
   NF_PUT_VAR       ! provide values for variable
     ...
NF_CLOSE            ! close: save new netCDF dataset
```

Only one call is needed to create a netCDF dataset, at which point you will be in the first of two netCDF modes. When accessing an open netCDF dataset, it is either in define mode or data mode. In define mode, you can create dimensions, variables, and new attributes, but you cannot read or write variable data. In data mode, you can access data and change existing attributes, but you are not permitted to create new dimensions, variables, or attributes.

One call to NF_DEF_DIM is needed for each dimension created. Similarly, one call to NF_DEF_VAR is needed for each variable creation, and one call to a member of the NF_PUT_ATT family is needed for each attribute defined and assigned a value. To leave define mode and enter data mode, call NF_ENDDEF.

Once in data mode, you can add new data to variables, change old values, and change values of existing attributes (so long as the attribute changes do not require more storage space). Single values may be written to a netCDF variable with one of the members of the NF_PUT_VAR1 family, depending on what type of data you have to write. All the values of a variable may be written at once with one of the members of the NF_PUT_VAR family. Arrays or array cross-sections of a variable may be written using members of the NF_PUT_VARA family. Subsampled array sections may be written using members of the NF_PUT_VARS family. Mapped array sections may be written using members of the NF_PUT_VARM family. (Subsampled and mapped access are general forms of data access that are explained later.)

Finally, you should explicitly close all netCDF datasets that have been opened for writing by calling NF_CLOSE. By default, access to the file system is buffered by the netCDF library. If a program terminates abnormally with netCDF datasets open for writing, your most recent modifications may be lost. This default buffering of data is disabled by setting the NF_SHARE flag when opening the dataset. But even if this flag is set, changes to attribute values or changes made in define mode are not written out until NF_SYNC or NF_CLOSE is called.

1.2 Reading a NetCDF Dataset with Known Names

Here we consider the case where you know the names of not only the netCDF datasets, but also the names of their dimensions, variables, and attributes. (Otherwise you would have to do "inquire" calls.) The order of typical C calls to read data from those variables in a netCDF dataset is:

```
NF_OPEN               ! open existing netCDF dataset
     ...
   NF_INQ_DIMID       ! get dimension IDs
     ...
   NF_INQ_VARID       ! get variable IDs
     ...
   NF_GET_ATT         ! get attribute values
     ...
   NF_GET_VAR         ! get values of variables
     ...
NF_CLOSE              ! close netCDF dataset
```

First, a single call opens the netCDF dataset, given the dataset name, and returns a netCDF ID that is used to refer to the open netCDF dataset in all subsequent calls.

Next, a call to NF_INQ_DIMID for each dimension of interest gets the dimension ID from the dimension name. Similarly, each required variable ID is determined from its name by a call to NF_INQ_VARID.Once variable IDs are known, variable attribute values can be retrieved using the netCDF ID, the variable ID, and the desired attribute name as input to a member of the NF_GET_ATT family (typically NF_GET_ATT_TEXT or NF_GET_ATT_DOUBLE) for each desired attribute. Variable data values can be directly accessed from the netCDF dataset with calls to members of the NF_GET_VAR1 family for single values, the NF_GET_VAR family for entire variables, or various other members of the NF_GET_VARA, NF_GET_VARS, or NF_GET_VARM families for array, subsampled or mapped access.

Finally, the netCDF dataset is closed with NF_CLOSE. There is no need to close a dataset open only for reading.

1.3 Reading a netCDF Dataset with Unknown Names

It is possible to write programs (e.g., generic software) which do such things as processing every variable, without needing to know in advance the names of these variables. Similarly, the names of dimensions and attributes may be unknown.

Names and other information about netCDF objects may be obtained from netCDF datasets by calling inquire functions. These return information about a whole netCDF dataset, a dimension, a variable, or an attribute. The following template illustrates how they are used:

```
     NF_OPEN                  ! open existing netCDF dataset
       ...
     NF_INQ                   ! find out what is in it
          ...
        NF_INQ_DIM            ! get dimension names, lengths
          ...
        NF_INQ_VAR            ! get variable names, types, shapes
             ...
           NF_INQ_ATTNAME     ! get attribute names
             ...
           NF_INQ_ATT         ! get attribute values
             ...
           NF_GET_ATT         ! get attribute values
             ...
        NF_GET_VAR            ! get values of variables
          ...
     NF_CLOSE                 ! close netCDF dataset
```

As in the previous example, a single call opens the existing netCDF dataset, returning a netCDF ID. This netCDF ID is given to the NF_INQ routine, which returns the number of dimensions, the number of variables, the number of global attributes, and the ID of the unlimited dimension, if there is one.

All the inquire functions are inexpensive to use and require no I/O, since the information they provide is stored in memory when a netCDF dataset is first opened.

Dimension IDs use consecutive integers, beginning at 1. Also dimensions, once created, cannot be deleted. Therefore, knowing the number of dimension IDs in a netCDF dataset means knowing all the dimension IDs: they are the integers 1, 2, 3, ... up to the number of dimensions. For each dimension ID, a call to the inquire function NF_INQ_DIM returns the dimension name and length.

Variable IDs are also assigned from consecutive integers 1, 2, 3, ... up to the number of variables. These can be used in NF_INQ_VAR calls to find out the names, types, shapes, and the number of attributes assigned to each variable.

Once the number of attributes for a variable is known, successive calls to NF_INQ_ATTNAME return the name for each attribute given the netCDF ID, variable ID, and attribute number. Armed with the attribute name, a call to NF_INQ_ATT returns its type and length. Given the type and length, you can allocate enough space to hold the attribute values. Then a call to a member of the NF_GET_ATT family returns the attribute values.

Once the IDs and shapes of netCDF variables are known, data values can be accessed by calling a member of the NF_GET_VAR1 family for single values, or members of the NF_GET_VAR, NF_GET_VARA, NF_GET_VARS, or NF_GET_VARM for various kinds of array access.

1.4 Adding New Dimensions, Variables, Attributes

An existing netCDF dataset can be extensively altered. New dimensions, variables, and attributes can be added or existing ones renamed, and existing attributes can be deleted. Existing dimensions, variables, and attributes can be renamed. The following code template lists a typical sequence of calls to add new netCDF components to an existing dataset:

```
NF_OPEN             ! open existing netCDF dataset
  ...
NF_REDEF            ! put it into define mode
    ...
  NF_DEF_DIM        ! define additional dimensions (if any)
    ...
  NF_DEF_VAR        ! define additional variables (if any)
    ...
  NF_PUT_ATT        ! define other attributes (if any)
    ...
NF_ENDDEF           ! check definitions, leave define mode
    ...
  NF_PUT_VAR        ! provide new variable values
    ...
NF_CLOSE            ! close netCDF dataset
```

A netCDF dataset is first opened by the NF_OPEN call. This call puts the open dataset in data mode, which means existing data values can be accessed and changed, existing attributes can be changed (so long as they do not grow), but nothing can be added. To add new netCDF dimensions, variables, or attributes you must enter define mode, by calling NF_REDEF.In define mode, call NF_DEF_DIM to define new dimensions, NF_DEF_VAR to define new variables, and a member of the NF_PUT_ATT family to assign new attributes to variables or enlarge old attributes.

You can leave define mode and reenter data mode, checking all the new definitions for consistency and committing the changes to disk, by calling NF_ENDDEF. If you do not wish to reenter data mode, just call NF_CLOSE, which will have the effect of first calling NF_ENDDEF.

Until the NF_ENDDEF call, you may back out of all the redefinitions made in define mode and restore the previous state of the netCDF dataset by calling NF_ABORT. You may also use the NF_ABORT call to restore the netCDF dataset to a consistent state if the call to NF_ENDDEF fails. If you have called NF_CLOSE from definition mode and the implied call to NF_ENDDEF fails, NF_ABORT will automatically be called to close the netCDF dataset and leave it in its previous consistent state (before you entered define mode).

At most one process should have a netCDF dataset open for writing at one time. The library is designed to provide limited support for multiple concurrent readers with one writer, via disciplined use of the NF_SYNC function and the NF_SHARE flag. If a writer makes changes in define mode, such as the addition of new variables, dimensions, or attributes, some means external to the library is necessary to prevent readers from making concurrent accesses and to inform readers to call NF_SYNC before the next access.

1.5 Error Handling

The netCDF library provides the facilities needed to handle errors in a flexible way. Each netCDF function returns an integer status value. If the returned status value indicates an error, you may handle it in any way desired, from printing an associated error message and exiting to ignoring the error indication and proceeding (not recommended!). For simplicity, the examples in this guide check the error status and call a separate function to handle any errors.

The NF_STRERROR function is available to convert a returned integer error status into an error message string.

Occasionally, low-level I/O errors may occur in a layer below the netCDF library. For example, if a write operation causes you to exceed disk quotas or to attempt to write to a device that is no longer available, you may get an error from a layer below the netCDF library, but the resulting write error will still be reflected in the returned status value.

1.6 Compiling and Linking with the NetCDF Library

Details of how to compile and link a program that uses the netCDF C or FORTRAN interfaces differ, depending on the operating system, the available compilers, and where the netCDF library and include files are installed. Nevertheless, we provide here examples of how to compile and link a program that uses the netCDF library on a Unix platform, so that you can adjust these examples to fit your installation.

Every FORTRAN file that references netCDF functions or constants must contain an appropriate INCLUDE statement before the first such reference:

```
INCLUDE 'netcdf.inc'
```

Unless the netcdf.inc file is installed in a standard directory where the FORTRAN compiler always looks, you must use the -I option when invoking the compiler, to specify a directory where netcdf.inc is installed, for example:

```
f77 -c -I/usr/local/netcdf/include myprogram.f
```

Alternatively, you could specify an absolute path name in the INCLUDE statement, but then your program would not compile on another platform where netCDF is installed in a different location.

Unless the netCDF library is installed in a standard directory where the linker always looks, you must use the -L and -l options to link an object file that uses the netCDF library. For example:

```
f77 -o myprogram myprogram.o -L/usr/local/netcdf/lib -lnetcdf
```

Alternatively, you could specify an absolute path name for the library:

```
f77 -o myprogram myprogram.o -l/usr/local/netcdf/lib/libnetcdf.
```

2 Datasets

2.1 Datasets Introduction

This chapter presents the interfaces of the netCDF functions that deal with a netCDF dataset or the whole netCDF library.

A netCDF dataset that has not yet been opened can only be referred to by its dataset name. Once a netCDF dataset is opened, it is referred to by a netCDF ID, which is a small nonnegative integer returned when you create or open the dataset. A netCDF ID is much like a file descriptor in C or a logical unit number in FORTRAN. In any single program, the netCDF IDs of distinct open netCDF datasets are distinct. A single netCDF dataset may be opened multiple times and will then have multiple distinct netCDF IDs; however at most one of the open instances of a single netCDF dataset should permit writing. When an open netCDF dataset is closed, the ID is no longer associated with a netCDF dataset.

Functions that deal with the netCDF library include:

- Get version of library.
- Get error message corresponding to a returned error code.

The operations supported on a netCDF dataset as a single object are:

- Create, given dataset name and whether to overwrite or not.
- Open for access, given dataset name and read or write intent.
- Put into define mode, to add dimensions, variables, or attributes.
- Take out of define mode, checking consistency of additions.
- Close, writing to disk if required.
- Inquire about the number of dimensions, number of variables, number of global attributes, and ID of the unlimited dimension, if any.
- Synchronize to disk to make sure it is current.
- Set and unset nofill mode for optimized sequential writes.
- After a summary of conventions used in describing the netCDF interfaces, the rest of this chapter presents a detailed description of the interfaces for these operations.

2.2 NetCDF Library Interface Descriptions

Each interface description for a particular netCDF function in this and later chapters contains:

- a description of the purpose of the function;
- a FORTRAN function prototype that presents the type and order of the formal parameters to the function;
- a description of each formal parameter in the C interface;
- a list of possible error conditions; and
- an example of a FORTRAN program fragment calling the netCDF function (and perhaps other netCDF functions).

The examples follow a simple convention for error handling, always checking the error status returned from each netCDF function call and calling a handle_error function in case an error was detected. For an example of such a function, see Section 5.2 "Get error message corresponding to error status: nf_strerror".

2.3 NF_STRERROR

The function NF_STRERROR returns a static reference to an error message string corresponding to an integer netCDF error status or to a system error number, presumably returned by a previous call to some other netCDF function. The list of netCDF error status codes is available in the appropriate include file for each language binding.

Usage

```
CHARACTER*80 FUNCTION NF_STRERROR(INTEGER NCERR)
```

`NCERR` An error status that might have been returned from a previous call to some netCDF function.

Errors

If you provide an invalid integer error status that does not correspond to any netCDF error message or or to any system error message (as understood by the system strerror function), NF_STRERROR returns a string indicating that there is no such error status.

Example

Here is an example of a simple error handling function that uses NF_STRERROR to print the error message corresponding to the netCDF error status returned from any netCDF function call and then exit:

```
INCLUDE 'netcdf.inc'
   ...
SUBROUTINE HANDLE_ERR(STATUS)
INTEGER STATUS
IF (STATUS .NE. NF_NOERR) THEN
  PRINT *, NF_STRERROR(STATUS)
  STOP 'Stopped'
ENDIF
END
```

2.4 Get netCDF library version: NF_INQ_LIBVERS

The function NF_INQ_LIBVERS returns a string identifying the version of the netCDF library, and when it was built.

Usage

```
CHARACTER*80 FUNCTION NF_INQ_LIBVERS()
```

Errors

This function takes no arguments, and thus no errors are possible in its invocation.

Example

Here is an example using nf_inq_libvers to print the version of the netCDF library with which the program is linked:

```
INCLUDE 'netcdf.inc'
   ...
PRINT *, NF_INQ_LIBVERS()
```

2.5 NF_CREATE

This function creates a new netCDF dataset, returning a netCDF ID that can subsequently be used to refer to the netCDF dataset in other netCDF function calls. The new netCDF dataset opened for write access and placed in define mode, ready for you to add dimensions, variables, and attributes.

A creation mode flag specifies whether to overwrite any existing dataset with the same name and whether access to the dataset is shared.

Usage

```
INTEGER FUNCTION NF_CREATE (CHARACTER*(*) PATH, INTEGER CMODE,
                            INTEGER ncid)
```

`PATH` The file name of the new netCDF dataset.

`CMODE` The creation mode flag. The following flags are available: NF_NOCLOBBER, NF_SHARE, NF_64BIT_OFFSET, NF_NETCDF4 and NF_CLASSIC_MODEL. You can combine the affect of multiple flags in a single argument by using the bitwise OR operator. For example, to specify both NF_NOCLOBBER and NF_SHARE, you could provide the argument OR(NF_NOCLOBBER, NF_SHARE).

A zero value (defined for convenience as NF_CLOBBER) specifies the default behavior: overwrite any existing dataset with the same file name and buffer and cache accesses for efficiency. The dataset will be in netCDF classic format. See Section "NetCDF Classic Format Limitations" in *The NetCDF Users Guide*.

Setting NF_NOCLOBBER means you do not want to clobber (overwrite) an existing dataset; an error (NF_EEXIST) is returned if the specified dataset already exists.

The NF_SHARE flag is appropriate when one process may be writing the dataset and one or more other processes reading the dataset concurrently; it means that dataset accesses are not buffered and caching is limited. Since the buffering scheme is optimized for sequential access, programs that do not access data sequentially may see some performance improvement by setting the NF_SHARE flag. This only applied to classic and 64-bit offset format files.

Setting NF_64BIT_OFFSET causes netCDF to create a 64-bit offset format file, instead of a netCDF classic format file. The 64-bit offset format imposes far fewer restrictions on very large (i.e. over 2 GB) data files. See Section "Large File Support" in *The NetCDF Users Guide*.

Setting NF_NETCDF4 causes netCDF to create a netCDF-4/HDF5 format file. Oring NF_CLASSIC_MODEL with NF_NETCDF4 causes the netCDF library to create a netCDF-4/HDF5 data file, with the netCDF classic model enforced - none of the new features of the netCDF-4 data model may be usedin such a file, for example groups and user-defined types.

`ncid` Returned netCDF ID.

Errors

NF_CREATE returns the value NF_NOERR if no errors occurred. Possible causes of errors include:

- Passing a dataset name that includes a directory that does not exist.
- Specifying a dataset name of a file that exists and also specifying NF_NOCLOBBER.
- Specifying a meaningless value for the creation mode.
- Attempting to create a netCDF dataset in a directory where you don't have permission to create files.

Example

In this example we create a netCDF dataset named foo.nc; we want the dataset to be created in the current directory only if a dataset with that name does not already exist:

```
INCLUDE 'netcdf.inc'
  ...
INTEGER NCID, STATUS
...
STATUS = NF_CREATE('foo.nc', NF_NOCLOBBER, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.6 NF__CREATE

This function is a variant of NF_CREATE, NF__CREATE (note the double underscore) allows users to specify two tuning parameters for the file that it is creating. These tuning parameters are not written to the data file, they are only used for so long as the file remains open after an NF__CREATE.

This function creates a new netCDF dataset, returning a netCDF ID that can subsequently be used to refer to the netCDF dataset in other netCDF function calls. The new netCDF dataset opened for write access and placed in define mode, ready for you to add dimensions, variables, and attributes.

A creation mode flag specifies whether to overwrite any existing dataset with the same name and whether access to the dataset is shared.

Usage

```
INTEGER FUNCTION NF__CREATE (CHARACTER*(*) PATH, INTEGER CMODE, INTEGER INITIALSZ,
                             INTEGER BUFRSIZEHINT, INTEGER ncid)
```

`PATH` The file name of the new netCDF dataset.

`CMODE` The creation mode flag. The following flags are available: NF_NOCLOBBER, NF_SHARE, NF_64BIT_OFFSET, NF_NETCDF4, and NF_CLASSIC_MODEL.

Setting NF_NOCLOBBER means you do not want to clobber (overwrite) an existing dataset; an error (NF_EEXIST) is returned if the specified dataset already exists.

The NF_SHARE flag is appropriate when one process may be writing the dataset and one or more other processes reading the dataset concurrently; it means that dataset accesses are not buffered and caching is limited. Since the buffering scheme is optimized for sequential access, programs that do not access data sequentially may see some performance improvement by setting the NF_SHARE flag. This flag has no effect with netCDF-4/HDF5 files.

Setting NF_64BIT_OFFSET causes netCDF to create a 64-bit offset format file, instead of a netCDF classic format file. The 64-bit offset format imposes far fewer restrictions on very large (i.e. over 2 GB) data files. See Section "Large File Support" in *The NetCDF Users Guide*.

Setting NF_CLASSIC_MODEL causes netCDF to enforce the classic data model in this file. (This only has effect for netCDF-4/HDF5 files, as classic and 64-bit offset files always use the classic model.) When used with NF_NETCDF4, this flag ensures that the resulting netCDF-4/HDF5 file may never contain any new constructs from the enhanced data model. That is, it cannot contain groups, user defined types, multiple unlimited dimensions, or new atomic types. The advantage of this restriction is that such files are guarenteed to work with existing netCDF software.

A zero value (defined for convenience as NF_CLOBBER) specifies the default behavior: overwrite any existing dataset with the same file name and buffer and cache accesses for efficiency. The dataset will be in netCDF classic format. See Section "NetCDF Classic Format Limitations" in *The NetCDF Users Guide*.

`INITIALSZ`
This parameter sets the initial size of the file at creation time.

`BUFRSIZEHINT`
The argument referenced by BUFRSIZEHINT controls a space versus time tradeoff, memory allocated in the netcdf library versus number of system calls.

Because of internal requirements, the value may not be set to exactly the value requested. The actual value chosen is returned by reference.

Using the value NF_SIZEHINT_DEFAULT causes the library to choose a default. How the system chooses the default depends on the system. On many systems, the "preferred I/O block size" is available from the stat() system call, struct stat member st_blksize. If this is available it is used. Lacking that, twice the system pagesize is used.

Lacking a call to discover the system pagesize, we just set default bufrsize to 8192.

The BUFRSIZE is a property of a given open netcdf descriptor ncid, it is not a persistent property of the netcdf dataset.

`ncid` Returned netCDF ID.

Errors

NF__CREATE returns the value NF_NOERR if no errors occurred. Possible causes of errors include:

- Passing a dataset name that includes a directory that does not exist.
- Specifying a dataset name of a file that exists and also specifying NF_NOCLOBBER.
- Specifying a meaningless value for the creation mode.
- Attempting to create a netCDF dataset in a directory where you don't have permission to create files.

Example

In this example we create a netCDF dataset named foo.nc; we want the dataset to be created in the current directory only if a dataset with that name does not already exist:

```
INCLUDE 'netcdf.inc'
  ...
INTEGER NCID, STATUS, INITIALSZ, BUFRSIZEHINT
...
INITIALSZ = 2048
BUFRSIZEHINT = 1024
STATUS = NF__CREATE('foo.nc', NF_NOCLOBBER, INITIALSZ, BUFRSIZEHINT, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.7 NF_CREATE_PAR

This function is a variant of nf_create, nf_create_par allows users to open a file on a MPI/IO or MPI/Posix parallel file system.

The parallel parameters are not written to the data file, they are only used for so long as the file remains open after an nf_create_par.

This function is only available if the netCDF library was built with parallel I/O.

This function creates a new netCDF dataset, returning a netCDF ID that can subsequently be used to refer to the netCDF dataset in other netCDF function calls. The new netCDF dataset opened for write access and placed in define mode, ready for you to add dimensions, variables, and attributes.

When a netCDF-4 file is created for parallel access, independent operations are the default. To use collective access on a variable, See Section 6.32 [NF_VAR_PAR_ACCESS], page 116.

Usage

```
INTEGER FUNCTION NF_CREATE_PAR(CHARACTER*(*) PATH, INTEGER CMODE,
                               INTEGER MPI_COMM, INTEGER MPI_INFO,
                               INTEGER ncid)
```

PATH The file name of the new netCDF dataset.

CMODE The creation mode flag. The following flags are available: NF_NOCLOBBER, NF_NETCDF4 and NF_CLASSIC_MODEL. You can combine the affect of multiple flags in a single argument by using the bitwise OR operator. For example, to specify both NF_NOCLOBBER and NF_NETCDF4, you could provide the argument OR(NF_NOCLOBBER, NF_NETCDF4).

Setting NF_NETCDF4 causes netCDF to create a netCDF-4/HDF5 format file. Oring NF_CLASSIC_MODEL with NF_NETCDF4 causes the netCDF library to create a netCDF-4/HDF5 data file, with the netCDF classic model enforced - none of the new features of the netCDF-4 data model may be usedin such a file, for example groups and user-defined types.

Only netCDF-4/HDF5 files may be used with parallel I/O.

MPI_COMM The MPI communicator.

MPI_INFO The MPI info.

ncid Returned netCDF ID.

Errors

NF_CREATE returns the value NF_NOERR if no errors occurred. Possible causes of errors include:

- Passing a dataset name that includes a directory that does not exist.
- Specifying a dataset name of a file that exists and also specifying NF_NOCLOBBER.
- Specifying a meaningless value for the creation mode.
- Attempting to create a netCDF dataset in a directory where you don't have permission to create files.

Example

This example is from test program nf_test/ftst_parallel.F.

```
!     Create the netCDF file.
      mode_flag = IOR(nf_netcdf4, nf_classic_model)
      retval = nf_create_par(FILE_NAME, mode_flag, MPI_COMM_WORLD,
     $     MPI_INFO_NULL, ncid)
      if (retval .ne. nf_noerr) stop 2
```

2.8 NF_OPEN

The function NF_OPEN opens an existing netCDF dataset for access.

Usage

```
INTEGER FUNCTION NF_OPEN(CHARACTER*(*) PATH, INTEGER OMODE, INTEGER ncid)
```

PATH File name for netCDF dataset to be opened. This may be an OPeNDAP URL if DAP support is enabled.

OMODE A zero value (or NF_NOWRITE) specifies the default behavior: open the dataset with read-only access, buffering and caching accesses for efficiency.

Otherwise, the creation mode is NF_WRITE, NF_SHARE, or OR(NF_WRITE, NF_SHARE). Setting the NF_WRITE flag opens the dataset with read-write access. ("Writing" means any kind of change to the dataset, including appending or changing data, adding or renaming dimensions, variables, and attributes, or deleting attributes.) The NF_SHARE flag is appropriate when one process may be writing the dataset and one or more other processes reading the dataset concurrently; it means that dataset accesses are not buffered and caching is limited. Since the buffering scheme is optimized for sequential access, programs that do not access data sequentially may see some performance improvement by setting the NF_SHARE flag.

ncid Returned netCDF ID.

Errors

NF_OPEN returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified netCDF dataset does not exist.
- A meaningless mode was specified.

Example

Here is an example using NF_OPEN to open an existing netCDF dataset named foo.nc for read-only, non-shared access:

```
INCLUDE 'netcdf.inc'
 ...
INTEGER NCID, STATUS
...
STATUS = NF_OPEN('foo.nc', 0, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.9 NF__OPEN

The function NF)_OPEN opens an existing netCDF dataset for access, with a performance tuning parameter.

Usage

```
INTEGER FUNCTION NF__OPEN(CHARACTER*(*) PATH, INTEGER OMODE, INTEGER
BUFRSIZEHINT, INTEGER ncid)
```

PATH File name for netCDF dataset to be opened.

OMODE A zero value (or NF_NOWRITE) specifies the default behavior: open the dataset with read-only access, buffering and caching accesses for efficiency

Otherwise, the creation mode is NF_WRITE, NF_SHARE, or OR(NF_WRITE,NF_SHARE). Setting the NF_WRITE flag opens the dataset with read-write access. ("Writing" means any kind of change to the dataset, including appending or changing data, adding or renaming dimensions, variables, and attributes, or deleting attributes.) The NF_SHARE flag is appropriate when one process may be writing the dataset and one or more other processes reading the dataset concurrently; it means that dataset accesses are not buffered and caching is limited. Since the buffering scheme is optimized for sequential access, programs that do not access data sequentially may see some performance improvement by setting the NF_SHARE flag.

`BUFRSIZEHINT`
: This argument controls a space versus time tradeoff, memory allocated in the netcdf library versus number of system calls.

  Because of internal requirements, the value may not be set to exactly the value requested. The actual value chosen is returned by reference.

  Using the value NF_SIZEHINT_DEFAULT causes the library to choose a default. How the system chooses the default depends on the system. On many systems, the "preferred I/O block size" is available from the stat() system call, struct stat member st_blksize. If this is available it is used. Lacking that, twice the system pagesize is used.

  Lacking a call to discover the system pagesize, we just set default bufrsize to 8192.

  The bufrsize is a property of a given open netcdf descriptor ncid, it is not a persistent property of the netcdf dataset.

`ncid`
: Returned netCDF ID.

Errors

NF__OPEN returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified netCDF dataset does not exist.
- A meaningless mode was specified.

Example

Here is an example using NF__OPEN to open an existing netCDF dataset named foo.nc for read-only, non-shared access:

```
INCLUDE 'netcdf.inc'
 ...
INTEGER NCID, STATUS, BUFRSIZEHINT
...
BUFRSIZEHINT = 1024
STATUS = NF_OPEN('foo.nc', 0, BUFRSIZEHINT, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.10 NF_OPEN_PAR

This function opens a netCDF-4 dataset for parallel access.

This function is only available if the netCDF library was built with a HDF5 library for which –enable-parallel was used, and which was linked (like HDF5) to MPI libraries.

This opens the file using either MPI-IO or MPI-POSIX. The file must be a netCDF-4 file. (That is, it must have been created using NF_NETCDF4 in the creation mode).

This function is only available if netCDF-4 was build with a version of the HDF5 library which was built with –enable-parallel.

Before either HDF5 or netCDF-4 can be installed with support for parallel programming, and MPI layer must also be installed on the machine, and usually a parallel file system.

NetCDF-4 exposes the parallel access functionality of HDF5. For more information about what is required to install and use the parallel access functions, see the HDF5 web site.

When a netCDF-4 file is opened for parallel access, collective operations are the default. To use independent access on a variable, See Section 6.32 [NF_VAR_PAR_ACCESS], page 116.

Usage

```
INTEGER FUNCTION NF_OPEN_PAR(CHARACTER*(*) PATH, INTEGER OMODE,
                             INTEGER MPI_COMM, INTEGER MPI_INFO,
                             INTEGER ncid)
```

`PATH` File name for netCDF dataset to be opened.

`OMODE` A zero value (or NF_NOWRITE) specifies the default behavior: open the dataset with read-only access.

Otherwise, the mode may be NF_WRITE. Setting the NF_WRITE flag opens the dataset with read-write access. ("Writing" means any kind of change to the dataset, including appending or changing data, adding or renaming dimensions, variables, and attributes, or deleting attributes.)

Setting NF_NETCDF4 is not necessary (or allowed). The file type is detected automatically.

`MPI_COMM` The MPI communicator.

`MPI_INFO` The MPI info.

`ncid` Returned netCDF ID.

Errors

NF_OPEN returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified netCDF dataset does not exist.
- A meaningless mode was specified.
- Not a netCDF-4 file.

Example

This example is from the test program nf_test/ftst_parallel.F.

```
!     Reopen the file.
      retval = nf_open_par(FILE_NAME, nf_nowrite, MPI_COMM_WORLD,
     $     MPI_INFO_NULL, ncid)
      if (retval .ne. nf_noerr) stop 2
```

2.11 NF_REDEF

The function NF_REDEF puts an open netCDF dataset into define mode, so dimensions, variables, and attributes can be added or renamed and attributes can be deleted.

Usage

```
INTEGER FUNCTION NF_REDEF(INTEGER NCID)
```

`NCID` netCDF ID, from a previous call to NF_OPEN or NF_CREATE.

Errors

NF_REDEF returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified netCDF dataset is already in define mode.
- The specified netCDF dataset was opened for read-only.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_REDEF to open an existing netCDF dataset named foo.nc and put it into define mode:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, STATUS
   ...
STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)   ! open dataset
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_REDEF(NCID)                      ! put in define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.12 NF_ENDDEF

The function NF_ENDDEF takes an open netCDF dataset out of define mode. The changes made to the netCDF dataset while it was in define mode are checked and committed to disk if no problems occurred. Non-record variables may be initialized to a "fill value" as well (see Section 2.18 [NF_SET_FILL], page 24). The netCDF dataset is then placed in data mode, so variable data can be read or written.

This call may involve copying data under some circumstances. See Section "File Structure and Performance" in *NetCDF Users' Guide*.

Usage

```
INTEGER FUNCTION NF_ENDDEF(INTEGER NCID)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

Errors

NF_ENDDEF returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified netCDF dataset is not in define mode.
- The specified netCDF ID does not refer to an open netCDF dataset. The size of one or more variables exceed the size constraints for whichever variant of the file format is in use). See Section "Large File Support" in *The NetCDF Users Guide*.
-

Example

Here is an example using NF_ENDDEF to finish the definitions of a new netCDF dataset named foo.nc and put it into data mode:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, STATUS
   ...
STATUS = NF_CREATE('foo.nc', NF_NOCLOBBER, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)

   ...   ! create dimensions, variables, attributes

STATUS = NF_ENDDEF(NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.13 NF__ENDDEF

The function NF__ENDDEF takes an open netCDF dataset out of define mode. The changes made to the netCDF dataset while it was in define mode are checked and committed to disk if no problems occurred. Non-record variables may be initialized to a "fill value" as well (see Section 2.18 [NF_SET_FILL], page 24). The netCDF dataset is then placed in data mode, so variable data can be read or written.

This call may involve copying data under some circumstances. See Section "File Structure and Performance" in *NetCDF Users' Guide*.

This function assumes specific characteristics of the netcdf version 1 and version 2 file formats. Users should use nf_enddef in most circumstances. Although this function will be available in future netCDF implementations, it may not continue to have any effect on performance.

The current netcdf file format has three sections, the "header" section, the data section for fixed size variables, and the data section for variables which have an unlimited dimension (record variables).

The header begins at the beginning of the file. The index (offset) of the beginning of the other two sections is contained in the header. Typically, there is no space between the sections. This causes copying overhead to accrue if one wishes to change the size of the sections, as may happen when changing names of things, text attribute values, adding attributes or adding variables. Also, for buffered i/o, there may be advantages to aligning sections in certain ways.

The minfree parameters allow one to control costs of future calls to nf_redef, nf_enddef by requesting that minfree bytes be available at the end of the section.

The align parameters allow one to set the alignment of the beginning of the corresponding sections. The beginning of the section is rounded up to an index which is a multiple of the align parameter. The flag value ALIGN_CHUNK tells the library to use the bufrsize (see above) as the align parameter.

The file format requires mod 4 alignment, so the align parameters are silently rounded up to multiples of 4. The usual call,

```
nf_enddef(ncid);
```

is equivalent to

```
nf_enddef(ncid, 0, 4, 0, 4);
```

The file format does not contain a "record size" value, this is calculated from the sizes of the record variables. This unfortunate fact prevents us from providing minfree and alignment control of the "records" in a netcdf file. If you add a variable which has an unlimited dimension, the third section will always be copied with the new variable added.

Usage

```
INTEGER FUNCTION NF_ENDDEF(INTEGER NCID, INTEGER H_MINFREE, INTEGER V_ALIGN,
               INTEGER V_MINFREE, INTEGER R_ALIGN)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`H_MINFREE`
Sets the pad at the end of the "header" section.

`V_ALIGN` Controls the alignment of the beginning of the data section for fixed size variables.

`V_MINFREE`
Sets the pad at the end of the data section for fixed size variables.

`R_ALIGN` Controls the alignment of the beginning of the data section for variables which have an unlimited dimension (record variables).

Errors

NF__ENDDEF returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified netCDF dataset is not in define mode.
- The specified netCDF ID does not refer to an open netCDF dataset.
- The size of one or more variables exceed the size constraints for whichever variant of the file format is in use). See Section "Large File Support" in *The NetCDF Users Guide*.

Example

Here is an example using NF__ENDDEF to finish the definitions of a new netCDF dataset named foo.nc and put it into data mode:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, STATUS, H_MINFREE, V_ALIGN, V_MINFREE, R_ALIGN
   ...
STATUS = NF_CREATE('foo.nc', NF_NOCLOBBER, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)

   ...   ! create dimensions, variables, attributes

H_MINFREE = 512
V_ALIGN = 512
V_MINFREE = 512
R_ALIGN = 512
STATUS = NF_ENDDEF(NCID, H_MINFREE, V_ALIGN, V_MINFREE, R_ALIGN)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.14 NF_CLOSE

The function NF_CLOSE closes an open netCDF dataset. If the dataset is in define mode, NF_ENDDEF will be called before closing. (In this case, if NF_ENDDEF returns an error, NF_ABORT will automatically be called to restore the dataset to the consistent state before define mode was last entered.) After an open netCDF dataset is closed, its netCDF ID may be reassigned to the next netCDF dataset that is opened or created.

Usage

```
INTEGER FUNCTION NF_CLOSE(INTEGER NCID)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

Errors

NF_CLOSE returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- Define mode was entered and the automatic call made to NF_ENDDEF failed.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_CLOSE to finish the definitions of a new netCDF dataset named foo.nc and release its netCDF ID:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, STATUS
   ...
```

```
STATUS = NF_CREATE('foo.nc', NF_NOCLOBBER, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)

   ...   ! create dimensions, variables, attributes

STATUS = NF_CLOSE(NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.15 NF_INQ Family

Members of the NF_INQ family of functions return information about an open netCDF dataset, given its netCDF ID. Dataset inquire functions may be called from either define mode or data mode. The first function, NF_INQ, returns values for the number of dimensions, the number of variables, the number of global attributes, and the dimension ID of the dimension defined with unlimited length, if any. The other functions in the family each return just one of these items of information.

For FORTRAN, these functions include NF_INQ, NF_INQ_NDIMS, NF_INQ_NVARS, NF_INQ_NATTS, and NF_INQ_UNLIMDIM. An additional function, NF_INQ_FORMAT, returns the (rarely needed) format version.

No I/O is performed when these functions are called, since the required information is available in memory for each open netCDF dataset.

Usage

```
INTEGER FUNCTION NF_INQ          (INTEGER NCID, INTEGER ndims,
                                  INTEGER nvars,INTEGER ngatts,
                                  INTEGER unlimdimid)
INTEGER FUNCTION NF_INQ_NDIMS    (INTEGER NCID, INTEGER ndims)
INTEGER FUNCTION NF_INQ_NVARS    (INTEGER NCID, INTEGER nvars)
INTEGER FUNCTION NF_INQ_NATTS    (INTEGER NCID, INTEGER ngatts)
INTEGER FUNCTION NF_INQ_UNLIMDIM (INTEGER NCID, INTEGER unlimdimid)
INTEGER FUNCTION NF_INQ_FORMAT   (INTEGER NCID, INTEGER format)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`ndims` Returned number of dimensions defined for this netCDF dataset.

`nvars` Returned number of variables defined for this netCDF dataset.

`ngatts` Returned number of global attributes defined for this netCDF dataset.

`unlimdimid`
Returned ID of the unlimited dimension, if there is one for this netCDF dataset. If no unlimited length dimension has been defined, -1 is returned.

`format` Returned format version, one of NF_FORMAT_CLASSIC, NF_FORMAT_64BIT, NF_FORMAT_NETCDF4, NF_FORMAT_NETCDF4_CLASSIC.

Errors

All members of the NF_INQ family return the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_INQ to find out about a netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID, NDIMS, NVARS, NGATTS, UNLIMDIMID
   ...
STATUS = NF_OPEN('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ(NCID, NDIMS, NVARS, NGATTS, UNLIMDIMID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.16 NF_SYNC

The function NF_SYNC offers a way to synchronize the disk copy of a netCDF dataset with in-memory buffers. There are two reasons you might want to synchronize after writes:

- To minimize data loss in case of abnormal termination, or
- To make data available to other processes for reading immediately after it is written. But note that a process that already had the dataset open for reading would not see the number of records increase when the writing process calls NF_SYNC; to accomplish this, the reading process must call NF_SYNC.

This function is backward-compatible with previous versions of the netCDF library. The intent was to allow sharing of a netCDF dataset among multiple readers and one writer, by having the writer call NF_SYNC after writing and the readers call NF_SYNC before each read. For a writer, this flushes buffers to disk. For a reader, it makes sure that the next read will be from disk rather than from previously cached buffers, so that the reader will see changes made by the writing process (e.g., the number of records written) without having to close and reopen the dataset. If you are only accessing a small amount of data, it can be expensive in computer resources to always synchronize to disk after every write, since you are giving up the benefits of buffering.

An easier way to accomplish sharing (and what is now recommended) is to have the writer and readers open the dataset with the NF_SHARE flag, and then it will not be necessary to call NF_SYNC at all. However, the NF_SYNC function still provides finer granularity than the NF_SHARE flag, if only a few netCDF accesses need to be synchronized among processes.

It is important to note that changes to the ancillary data, such as attribute values, are not propagated automatically by use of the NF_SHARE flag. Use of the NF_SYNC function is still required for this purpose.

Sharing datasets when the writer enters define mode to change the data schema requires extra care. In previous releases, after the writer left define mode, the readers were left looking at an old copy of the dataset, since the changes were made to a new copy. The only way readers could see the changes was by closing and reopening the dataset. Now the changes are made in place, but readers have no knowledge that their internal tables are now inconsistent with the new dataset schema. If netCDF datasets are shared across redefinition, some mechanism external to the netCDF library must be provided that prevents access by readers during redefinition and causes the readers to call NF_SYNC before any subsequent access.

When calling NF_SYNC, the netCDF dataset must be in data mode. A netCDF dataset in define mode is synchronized to disk only when NF_ENDDEF is called. A process that is reading a netCDF dataset that another process is writing may call NF_SYNC to get updated with the changes made to the data by the writing process (e.g., the number of records written), without having to close and reopen the dataset.

Data is automatically synchronized to disk when a netCDF dataset is closed, or whenever you leave define mode.

Usage

```
INTEGER FUNCTION NF_SYNC(INTEGER NCID)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

Errors

NF_SYNC returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The netCDF dataset is in define mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_SYNC to synchronize the disk writes of a netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID
   ...
STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! write data or change attributes
   ...
STATUS = NF_SYNC(NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

2.17 NF_ABORT

You no longer need to call this function, since it is called automatically by NF_CLOSE in case the dataset is in define mode and something goes wrong with committing the changes.

The function NF_ABORT just closes the netCDF dataset, if not in define mode. If the dataset is being created and is still in define mode, the dataset is deleted. If define mode was entered by a call to NF_REDEF, the netCDF dataset is restored to its state before definition mode was entered and the dataset is closed.

Usage

```
INTEGER FUNCTION NF_ABORT(INTEGER NCID)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

Errors

NF_ABORT returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- When called from define mode while creating a netCDF dataset, deletion of the dataset failed.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_ABORT to back out of redefinitions of a dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID, LATID
   ...
STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_REDEF(NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_DEF_DIM(NCID, 'LAT', 18, LATID)
IF (STATUS .NE. NF_NOERR) THEN  ! dimension definition failed
   CALL HANDLE_ERR(STATUS)
   STATUS = NF_ABORT(NCID)  ! abort redefinitions
   IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
ENDIF
...
```

2.18 NF_SET_FILL

This function is intended for advanced usage, to optimize writes under some circumstances described below. The function NF_SET_FILL sets the fill mode for a netCDF dataset open for writing and returns the current fill mode in a return parameter. The fill mode can be specified as either NF_FILL or NF_NOFILL. The default behavior corresponding to NF_FILL is that data is pre-filled with fill values, that is fill values are written when you create non-record variables or when you write a value beyond data that has not yet been written. This makes it possible to detect attempts to read data before it was written. See Section 6.30 [Fill Values], page 114, for more information on the use of fill values. See Section "Attribute Conventions" in *The NetCDF Users Guide*, for information about how to define your own fill values.

The behavior corresponding to NF_NOFILL overrides the default behavior of prefilling data with fill values. This can be used to enhance performance, because it avoids the duplicate writes that occur when the netCDF library writes fill values that are later overwritten with data.

A value indicating which mode the netCDF dataset was already in is returned. You can use this value to temporarily change the fill mode of an open netCDF dataset and then restore it to the previous mode.

After you turn on NF_NOFILL mode for an open netCDF dataset, you must be certain to write valid data in all the positions that will later be read. Note that nofill mode is only a transient property of a netCDF dataset open for writing: if you close and reopen the dataset, it will revert to the default behavior. You can also revert to the default behavior by calling NF_SET_FILL again to explicitly set the fill mode to NF_FILL.

There are three situations where it is advantageous to set nofill mode:

1. Creating and initializing a netCDF dataset. In this case, you should set nofill mode before calling NF_ENDDEF and then write completely all non-record variables and the initial records of all the record variables you want to initialize.
2. Extending an existing record-oriented netCDF dataset. Set nofill mode after opening the dataset for writing, then append the additional records to the dataset completely, leaving no intervening unwritten records.
3. Adding new variables that you are going to initialize to an existing netCDF dataset. Set nofill mode before calling NF_ENDDEF then write all the new variables completely.

If the netCDF dataset has an unlimited dimension and the last record was written while in nofill mode, then the dataset may be shorter than if nofill mode was not set, but this will be completely transparent if you access the data only through the netCDF interfaces.

The use of this feature may not be available (or even needed) in future releases. Programmers are cautioned against heavy reliance upon this feature.

Usage

```
INTEGER FUNCTION NF_SET_FILL(INTEGER NCID, INTEGER FILLMODE,
                             INTEGER old_mode)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`FILLMODE` Desired fill mode for the dataset, either NF_NOFILL or NF_FILL.

`old_mode` Returned current fill mode of the dataset before this call, either NF_NOFILL or NF_FILL.

Errors

NF_SET_FILL returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified netCDF ID does not refer to an open netCDF dataset.
- The specified netCDF ID refers to a dataset open for read-only access.
- The fill mode argument is neither NF_NOFILL nor NF_FILL..

Example

Here is an example using NF_SET_FILL to set nofill mode for subsequent writes of a netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, STATUS, OMODE
   ...
STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! write data with default prefilling behavior
   ...
STATUS = NF_SET_FILL(NCID, NF_NOFILL, OMODE)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! write data with no prefilling
   ...
```

2.19 NF_SET_DEFAULT_FORMAT

This function is intended for advanced users.

In version 3.6, netCDF introduced a new data format, the first change in the underlying binary data format since the netCDF interface was released. The new format, 64-bit offset format, was introduced to greatly relax the limitations on creating very large files.

In version 4.0, another new binary format was introduced: netCDF-4/HDF5.

Users are warned that creating files in the 64-bit offset format makes them unreadable by the netCDF library prior to version 3.6.0, and creating files in netcdf-4/HDF5 format makes them unreadable by the netCDF library prior to version 4.0. For reasons of compatibility, users should continue to create files in netCDF classic format.

Users who do want to use 64-bit offset or netCDF-4/HDF5 format files can create them directory from NF_CREATE, using the proper cmode flag. (see Section 2.5 [NF_CREATE], page 9).

The function NF_SET_DEFAULT_FORMAT allows the user to change the format of the netCDF file to be created by future calls to NF_CREATE without changing the cmode flag.

This allows the user to convert a program to use the new formats without changing all calls the NF_CREATE.

Once the default format is set, all future created files will be in the desired format.

Constants are provided in the netcdf.inc file to be used with this function: nf_format_classic, nf_format_64bit, nf_format_netcdf4 and nf_format_netcdf4_classic.

Usage

```
INTEGER FUNCTION NF_SET_DEFAULT_FORMAT(INTEGER FORMAT, INTEGER OLD_FORMT)
```

`FORMAT` Either nf_format_classic, nf_format_64bit, nf_format_netcdf4 or nf_format_netcdf4_classic.

`OLD_FORMAT`
The default format at the time the function is called is returned here.

Errors

The following error codes may be returned by this function:

- An NF_EINVAL error is returned if an invalid default format is specified.

Example

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, OLD_FORMAT
   ...
STATUS = NF_SET_DEFAULT_FORMAT(nf_format_64bit, OLD_FORMAT)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
```

2.20 Set HDF5 Chunk Cache for Future File Opens/Creates: NF_SET_CHUNK_CACHE

This function changes the chunk cache settings in the HDF5 library. The settings apply for subsequent file opens/creates. This function does not change the chunk cache settings of already open files.

This affects the per-file chunk cache which the HDF5 layer maintains. The chunk cache size can be tuned for better performance.

For more information, see the documentation for the H5Pset_cache() function in the HDF5 library at the HDF5 website: `http://hdfgroup.org/HDF5/`.

Usage

```
INTEGER NF_SET_CHUNK_CACHE(INTEGER SIZE, INTEGER NELEMS, INTEGER PREEMPTION);
```

`SIZE` The total size of the raw data chunk cache in MegaBytes.

`NELEMS` The number slots in the per-variable chunk cache (should be a prime number larger than the number of chunks in the cache).

`PREEMPTION`
The preemtion value must be between 0 and 100 inclusive and indicates how much chunks that have been fully read are favored for preemption. A value of zero means fully read chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 100 means fully read chunks are always preempted before other chunks.

Return Codes

`NF_NOERR` No error.

`NF_EINVAL`
: Parameters size and nelems must be non-zero positive integers, and preemption must be between zero and 100 (inclusive). An NF_EINVAL will be returned otherwise.

2.21 Get the HDF5 Chunk Cache Settings for Future File Opens/Creates: NF_GET_CHUNK_CACHE

This function gets the chunk cache settings for the HDF5 library. The settings apply for subsequent file opens/creates.

This affects the per-file chunk cache which the HDF5 layer maintains. The chunk cache size can be tuned for better performance.

For more information, see the documentation for the H5Pget_cache() function in the HDF5 library at the HDF5 website: `http://hdfgroup.org/HDF5/`.

Usage

```
INTEGER NC_GET_CHUNK_CACHE(INTEGER SIZE, INTEGER NELEMS, INTEGER PREEMPTION);
```

`SIZE` The total size of the raw data chunk cache will be put here.

`NELEMS` The number of chunk slots in the raw data chunk cache hash table will be put here.

`PREEMPTION`
: The preemption will be put here. The preemtion value is between 0 and 100 inclusive and indicates how much chunks that have been fully read are favored for preemption. A value of zero means fully read chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 100 means fully read chunks are always preempted before other chunks.

Return Codes

`NC_NOERR` No error.

3 Groups

NetCDF-4 added support for hierarchical groups within netCDF datasets.

Groups are identified with a ncid, which identifies both the open file, and the group within that file. When a file is opened with NF_OPEN or NF_CREATE, the ncid for the root group of that file is provided. Using that as a starting point, users can add new groups, or list and navigate existing groups.

All netCDF calls take a ncid which determines where the call will take its action. For example, the NF_DEF_VAR function takes a ncid as its first parameter. It will create a variable in whichever group its ncid refers to. Use the root ncid provided by NF_CREATE or NF_OPEN to create a variable in the root group. Or use NF_DEF_GRP to create a group and use its ncid to define a variable in the new group.

Variable are only visible in the group in which they are defined. The same applies to attributes. "Global" attributes are defined in whichever group is refered to by the ncid.

Dimensions are visible in their groups, and all child groups.

Group operations are only permitted on netCDF-4 files - that is, files created with the HDF5 flag in nf_create. (see Section 2.5 [NF_CREATE], page 9). Groups are not compatible with the netCDF classic data model, so files created with the NF_CLASSIC_MODEL file cannot contain groups (except the root group).

3.1 Find a Group ID: NF_INQ_NCID

Given an ncid and group name (NULL or "" gets root group), return ncid of the named group.

Usage

```
INTEGER FUNCTION NF_INQ_NCID(INTEGER NCID, CHARACTER*(*) NAME, INTEGER GRPID)
```

`NCID` The group id for this operation.

`NAME` A character array that holds the name of the desired group. Must be less then NF_MAX_NAME.

`GRPID` The ID of the group will go here.

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
: An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     Check getting the group by name
      retval = nf_inq_ncid(ncid, group_name, grpid_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

3.2 Get a List of Groups in a Group: NF_INQ_GRPS

Given a location id, return the number of groups it contains, and an array of their ncids.

Usage

```
INTEGER FUNCTION NF_INQ_GRPS(INTEGER NCID, INTEGER NUMGRPS, INTEGER NCIDS)
```

`NCID` The group id for this operation.

`NUMGRPS` An integer which will get number of groups in this group.

`NCIDS` An array of ints which will receive the IDs of all the groups in this group.

Errors

`NF_NOERR` No error.

`NF_EBADID`
: Bad group id.

`NF_ENOTNC4`
: Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
: This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
: An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     What groups are there from the root group?
      retval = nf_inq_grps(ncid, ngroups_in, grpids)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

3.3 Find all the Variables in a Group: NF_INQ_VARIDS

Find all varids for a location.

Usage

```
INTEGER FUNCTION NF_INQ_VARIDS(INTEGER NCID, INTEGERS VARIDS)
```

`NCID` The group id for this operation.

`VARIDS` An already allocated array to store the list of varids. Use nf_inq_nvars to find out how many variables there are. (see Section 2.15 [NF_INQ Family], page 21).

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     Check varids in subgroup.
      retval = nf_inq_varids(subgrp_in, nvars, varids_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

3.4 Find all Dimensions Visible in a Group: NF_INQ_DIMIDS

Find all dimids for a location. This finds all dimensions in a group, or any of its parents.

Usage

```
INTEGER FUNCTION NF_INQ_DIMIDS(INTEGER NCID, INTEGER DIMIDS, INTEGER INCLUDE_PARENTS)
```

`NCID` The group id for this operation.

`DIMIDS` An array of ints when the dimids of the visible dimensions will be stashed. Use nf_inq_ndims to find out how many dims are visible from this group. (see Section 2.15 [NF_INQ Family], page 21).

`INCLUDE_PARENTS`
If zero, only the group specified by NCID will be searched for dimensions. Otherwise parent groups will be searched too.

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     Check dimids in subgroup.
      retval = nf_inq_dimids(subgrp_in, ndims, dimids_in, 0)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (ndims .ne. 2 .or. dimids_in(1) .ne. dimids(1) .or.
     &     dimids_in(2) .ne. dimids(2)) stop 2
```

3.5 Find the Length of a Group's Name: NF_INQ_GRPNAME_LEN

Given ncid, find length of the full name. (Root group is named "/", with length 1.)

Usage

```
INTEGER FUNCTION NF_INQ_GRPNAME_LEN(INTEGER NCID, INTEGER LEN)
```

`NCID` The group id for this operation.

`LEN` An integer where the length will be placed.

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

NF_ESTRICTNC3
: This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

NF_EHDFERR
: An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     Check the length of the full name.
      retval = nf_inq_grpname_len(grpids(1), full_name_len)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

3.6 Find a Group's Name: NF_INQ_GRPNAME

Given ncid, find relative name of group. (Root group is named "/").

The name provided by this function is relative to the parent group. For a full path name for the group is, with all parent groups included, separated with a forward slash (as in Unix directory names) See Section 3.7 [NF_INQ_GRPNAME_FULL], page 34.

Usage

```
INTEGER FUNCTION NF_INQ_GRPNAME(INTEGER NCID, CHARACTER*(*) NAME)
```

NCID
: The group id for this operation.

NAME
: The name of the group will be copied to this character array. The name will be less than NF_MAX_NAME in length.

Errors

NF_NOERR
: No error.

NF_EBADID
: Bad group id.

NF_ENOTNC4
: Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

NF_ESTRICTNC3
: This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

NF_EHDFERR
: An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     Check the name of the root group.
      retval = nf_inq_grpname(ncid, name_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (name_in(1:1) .ne. '/') stop 2
```

3.7 Find a Group's Full Name: NF_INQ_GRPNAME_FULL

Given ncid, find complete name of group. (Root group is named "/").

The name provided by this function is a full path name for the group is, with all parent groups included, separated with a forward slash (as in Unix directory names). For a name relative to the parent group See Section 3.6 [NF_INQ_GRPNAME], page 33.

To find the length of the full name See Section 3.5 [NF_INQ_GRPNAME_LEN], page 32.

Usage

```
INTEGER FUNCTION NF_INQ_GRPNAME_FULL(INTEGER NCID, INTEGER LEN, CHARACTER*(*) NAME)
```

`NCID` The group id for this operation.

`LEN` The length of the full group name will go here.

`NAME` The name of the group will be copied to this character array.

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     Check the full name.
      retval = nf_inq_grpname_full(grpids(1), full_name_len, name_in2)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

3.8 Find a Group's Parent: NF_INQ_GRP_PARENT

Given ncid, find the ncid of the parent group.

When used with the root group, this function returns the NF_ENOGRP error (since the root group has no parent.)

Usage

```
INTEGER FUNCTION NF_INQ_GRP_PARENT(INTEGER NCID, INTEGER PARENT_NCID)
```

`NCID` The group id.

`PARENT_NCID`
The ncid of the parent group will be copied here.

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENOGRP`
No parent group found (i.e. this is the root group).

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     Check the parent ncid.
      retval = nf_inq_grp_parent(grpids(1), grpid_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

3.9 Find a Group by Name: NF_INQ_GRP_NCID

Given a group name an an ncid, find the ncid of the group id.

Usage

```
INTEGER FUNCTION NF_INQ_GRP_NCID(INTEGER NCID, CHARACTER GRP_NAME, INTEGER GRP_NCID)
```

`NCID` The group id to look in.

`GRP_NAME` The name of the group that should be found.

`GRP_NCID` This will get the group id, if it is found.

Return Codes

The following return codes may be returned by this function.

`NF_NOERR` No error.

`NF_EBADID`
: Bad group id.

`NF_EINVAL`
: No name provided or name longer than NF_MAX_NAME.

`NF_ENOGRP`
: Named group not found.

`NF_ENOTNC4`
: Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
: This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
: An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_types3.F.

```
C     Go to a child group and find the id of our type.
      retval = nf_inq_grp_ncid(ncid, group_name, sub_grpid)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

3.10 Find a Group by its Fully-qualified Name: NF_INQ_GRP_FULL_NCID

Given a fully qualified group name an an ncid, find the ncid of the group id.

Usage

```
INTEGER FUNCTION NF_INQ_GRP_FULL_NCID(INTEGER NCID, CHARACTER FULL_NAME, INTEGER GRP_NCID)
```

`NCID` The group id to look in.

`FULL_NAME`
: The fully-qualified group name.

`GRP_NCID` This will get the group id, if it is found.

Return Codes

The following return codes may be returned by this function.

NF_NOERR
: No error.

NF_EBADID
: Bad group id.

NF_EINVAL
: No name provided or name longer than NF_MAX_NAME.

NF_ENOGRP
: Named group not found.

NF_ENOTNC4
: Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

NF_ESTRICTNC3
: This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

NF_EHDFERR
: An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_groups.F.

```
C     Check the full name of the root group (also "/").
      retval = nf_inq_grpname_full(ncid, full_name_len, name_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

3.11 Create a New Group: NF_DEF_GRP

Create a group. Its location id is returned in new_ncid.

Usage

```
INTEGER FUNCTION NF_DEF_GRP(INTEGER PARENT_NCID, CHARACTER*(*) NAME,
        INTEGER NEW_NCID)
```

PARENT_NCID
: The group id of the parent group.

NAME
: The name of the new group, which must be different from the name of any variable within the same parent group.

NEW_NCID
: The ncid of the new group will be placed there.

Errors

NF_NOERR
: No error.

NF_EBADID
: Bad group id.

NF_ENAMEINUSE
: That name is in use. Group names must be unique within a group.

NF_EMAXNAME
: Name exceed max length NF_MAX_NAME.

NF_EBADNAME
: Name contains illegal characters.

NF_ENOTNC4
: Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag HDF5. (see Section 2.8 [NF_OPEN], page 13).

NF_ESTRICTNC3
: This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

NF_EHDFERR
: An error was reported by the HDF5 layer.

NF_EPERM
: Attempt to write to a read-only file.

NF_ENOTINDEFINE
: Not in define mode.

Example

In this exampe rom nf_test/ftst_groups.F, a groups is reated, and then a sub-group is created in that group.

```
C     Create the netCDF file.
      retval = nf_create(file_name, NF_NETCDF4, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Create a group and a subgroup.
      retval = nf_def_grp(ncid, group_name, grpid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_grp(grpid, sub_group_name, sub_grpid)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

4 Dimensions

4.1 Dimensions Introduction

Dimensions for a netCDF dataset are defined when it is created, while the netCDF dataset is in define mode. Additional dimensions may be added later by reentering define mode. A netCDF dimension has a name and a length. At most one dimension in a netCDF dataset can have the unlimited length, which means variables using this dimension can grow along this dimension.

There is a suggested limit (100) to the number of dimensions that can be defined in a single netCDF dataset. The limit is the value of the predefined macro NF_MAX_DIMS. The purpose of the limit is to make writing generic applications simpler. They need only provide an array of NF_MAX_DIMS dimensions to handle any netCDF dataset. The implementation of the netCDF library does not enforce this advisory maximum, so it is possible to use more dimensions, if necessary, but netCDF utilities that assume the advisory maximums may not be able to handle the resulting netCDF datasets.

Ordinarily, the name and length of a dimension are fixed when the dimension is first defined. The name may be changed later, but the length of a dimension (other than the unlimited dimension) cannot be changed without copying all the data to a new netCDF dataset with a redefined dimension length.

A netCDF dimension in an open netCDF dataset is referred to by a small integer called a dimension ID. In the FORTRAN interface, dimension IDs are 1, 2, 3, ..., in the order in which the dimensions were defined.

Operations supported on dimensions are:

- Create a dimension, given its name and length.
- Get a dimension ID from its name.
- Get a dimension's name and length from its ID.
- Rename a dimension.

4.2 NF_DEF_DIM

The function NF_DEF_DIM adds a new dimension to an open netCDF dataset in define mode. It returns (as an argument) a dimension ID, given the netCDF ID, the dimension name, and the dimension length. At most one unlimited length dimension, called the record dimension, may be defined for each netCDF dataset.

Usage

```
INTEGER FUNCTION NF_DEF_DIM (INTEGER NCID, CHARACTER*(*) NAME,
                             INTEGER LEN, INTEGER dimid)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`NAME` Dimension name.

`LEN` Length of dimension; that is, number of values for this dimension as an index to variables that use it. This should be either a positive integer or the predefined constant NF_UNLIMITED.

`dimid` Returned dimension ID.

Errors

NF_DEF_DIM returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The netCDF dataset is not in definition mode.
- The specified dimension name is the name of another existing dimension.
- The specified length is not greater than zero.
- The specified length is unlimited, but there is already an unlimited length dimension defined for this netCDF dataset.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_DEF_DIM to create a dimension named lat of length 18 and a unlimited dimension named rec in a new netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID, LATID, RECID
   ...
STATUS = NF_CREATE('foo.nc', NF_NOCLOBBER, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_DEF_DIM(NCID, 'lat', 18, LATID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_DEF_DIM(NCID, 'rec', NF_UNLIMITED, RECID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

4.3 NF_INQ_DIMID

The function NF_INQ_DIMID returns (as an argument) the ID of a netCDF dimension, given the name of the dimension. If ndims is the number of dimensions defined for a netCDF dataset, each dimension has an ID between 1 and ndims.

Usage

```
INTEGER FUNCTION NF_INQ_DIMID (INTEGER NCID, CHARACTER*(*) NAME,
                               INTEGER dimid)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`NAME` Dimension name.

`dimid` Returned dimension ID.

Errors

NF_INQ_DIMID returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The name that was specified is not the name of a dimension in the netCDF dataset.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_INQ_DIMID to determine the dimension ID of a dimension named lat, assumed to have been defined previously in an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID, LATID
   ...
STATUS = NF_OPEN('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_DIMID(NCID, 'lat', LATID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

4.4 NF_INQ_DIM Family

This family of functions returns information about a netCDF dimension. Information about a dimension includes its name and its length. The length for the unlimited dimension, if any, is the number of records written so far.

The functions in this family include NF_INQ_DIM, NF_INQ_DIMNAME, and NF_INQ_DIMLEN. The function NF_INQ_DIM returns all the information about a dimension; the other functions each return just one item of information.

Usage

```
INTEGER FUNCTION NF_INQ_DIM     (INTEGER NCID, INTEGER DIMID,
                                 CHARACTER*(*) name, INTEGER len)
INTEGER FUNCTION NF_INQ_DIMNAME (INTEGER NCID, INTEGER DIMID,
                                 CHARACTER*(*) name)
INTEGER FUNCTION NF_INQ_DIMLEN  (INTEGER NCID, INTEGER DIMID,
                                 INTEGER len)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`DIMID` Dimension ID, from a previous call to NF_INQ_DIMID or NF_DEF_DIM.

`NAME` Returned dimension name. The caller must allocate space for the returned name. The maximum possible length, in characters, of a dimension name is given by the predefined constant NF_MAX_NAME.

`len` Returned length of dimension. For the unlimited dimension, this is the current maximum value used for writing any variables with this dimension, that is the maximum record number.

Errors

These functions return the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The dimension ID is invalid for the specified netCDF dataset.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_INQ_DIM to determine the length of a dimension named lat, and the name and current maximum length of the unlimited dimension for an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID, LATID, LATLEN, RECID, NRECS
CHARACTER*(NF_MAX_NAME) LATNAM, RECNAM
   ...
STATUS = NF_OPEN('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
! get ID of unlimited dimension
STATUS = NF_INQ_UNLIMDIM(NCID, RECID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_DIMID(NCID, 'lat', LATID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
! get lat length
STATUS = NF_INQ_DIMLEN(NCID, LATID, LATLEN)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
! get unlimited dimension name and current length
STATUS = NF_INQ_DIM(NCID, RECID, RECNAME, NRECS)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

4.5 NF_RENAME_DIM

The function NF_RENAME_DIM renames an existing dimension in a netCDF dataset open for writing. If the new name is longer than the old name, the netCDF dataset must be in define mode. You cannot rename a dimension to have the same name as another dimension.

Usage

```
INTEGER FUNCTION NF_RENAME_DIM (INTEGER NCID, INTEGER DIMID,
                                CHARACTER*(*) NAME)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`DIMID` Dimension ID, from a previous call to NF_INQ_DIMID or NF_DEF_DIM.

`NAME` New dimension name.

Errors

NF_RENAME_DIM returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The new name is the name of another dimension.
- The dimension ID is invalid for the specified netCDF dataset.
- The specified netCDF ID does not refer to an open netCDF dataset.
- The new name is longer than the old name and the netCDF dataset is not in define mode.

Example

Here is an example using NF_RENAME_DIM to rename the dimension lat to latitude in an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID, LATID
   ...
STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! put in define mode to rename dimension
STATUS = NF_REDEF(NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_INQ_DIMID(NCID, 'lat', LATID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_RENAME_DIM(NCID, LATID, 'latitude')
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
! leave define mode
STATUS = NF_ENDDEF(NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

5 User Defined Data Types

5.1 User Defined Types Introduction

NetCDF-4 has added support for four different user defined data types.

`compound type`
: Like a C struct, a compound type is a collection of types, including other user defined types, in one package.

`variable length array type`
: The variable length array may be used to store ragged arrays.

`opaque type`
: This type has only a size per element, and no other type information.

`enum type`
: Like an enumeration in C, this type lets you assign text values to integer values, and store the integer values.

Users may construct user defined type with the various NF_DEF_* functions described in this section. They may learn about user defined types by using the NF_INQ_ functions defined in this section.

Once types are constructed, define variables of the new type with NF_DEF_VAR (see Section 6.3 [NF_DEF_VAR], page 70). Write to them with NF_PUT_VAR1, NF_PUT_VAR, NF_PUT_VARA, or NF_PUT_VARS (see Chapter 6 [Variables], page 69). Read data of user-defined type with NF_GET_VAR1, NF_GET_VAR, NF_GET_VARA, or NF_GET_VARS (see Chapter 6 [Variables], page 69).

Create attributes of the new type with NF_PUT_ATT (see Section 7.2 [NF_PUT_ATT_ type], page 119). Read attributes of the new type with NF_GET_ATT (see Section 7.4 [NF_GET_ATT_ type], page 123).

5.2 Learn the IDs of All Types in Group: NF_INQ_TYPEIDS

Learn the number of types defined in a group, and their IDs.

Usage

```
INTEGER FUNCTION NF_INQ_TYPEIDS(INTEGER NCID, INTEGER NTYPES,
        INTEGER TYPEIDS)
```

`NCID`
: The group id.

`NTYPES`
: A pointer to int which will get the number of types defined in the group. If NULL, ignored.

`TYPEIDS`
: A pointer to an int array which will get the typeids. If NULL, ignored.

Errors

`NF_NOERR`
: No error.

`NF_BADID`
: Bad ncid.

Example

The following example is from the test program nf_test/ftst_vars3.F.

```
          retval = nf_inq_typeids(ncid, num_types, typeids)
          if (retval .ne. nf_noerr) call handle_err(retval)
```

5.3 Find a Typeid from Group and Name: NF_INQ_TYPEID

Given a group ID and a type name, find the ID of the type. If the type is not found in the group, then the parents are searched. If still not found, the entire file is searched.

Usage

```
INTEGER FUNCTION NF_INQ_TYPEID(INTEGER NCID, CHARACTER NAME, NF_TYPE TYPEIDP)
```

`NCID` The group id.

`NAME` The name of a type.

`TYPEIDP` The typeid of the named type (if found).

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad ncid.

`NF_EBADTYPE`
Can't find type.

Example

The following example is from nf_test/ftst_types3.F:

```
C     Go to a child group and find the id of our type.
      retval = nf_inq_grp_ncid(ncid, group_name, sub_grpid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_inq_typeid(sub_grpid, type_name, typeid_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.4 Learn About a User Defined Type: NF_INQ_TYPE

Given an ncid and a typeid, get the information about a type. This function will work on any type, including atomic and any user defined type, whether compound, opaque, enumeration, or variable length array.

For even more information about a user defined type Section 5.5 [NF_INQ_USER_TYPE], page 48.

Usage

```
INTEGER FUNCTION NF_INQ_TYPE(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) NAME, INTEGER SIZE)
```

`NCID` The ncid for the group containing the type (ignored for atomic types).

`XTYPE` The typeid for this type, as returned by NF_DEF_COMPOUND, NF_DEF_OPAQUE, NF_DEF_ENUM, NF_DEF_VLEN, or NF_INQ_VAR, or as found in netcdf.inc in the list of atomic types (NF_CHAR, NF_INT, etc.).

`NAME` The name of the user defined type will be copied here. It will be NF_MAX_NAME bytes or less. For atomic types, the type name from CDL will be given.

`SIZEP` The (in-memory) size of the type (in bytes) will be copied here. VLEN type size is the size of one vlen sturture (i.e. the sice of nc_vlen_t). String size is returned as the size of one C character pointer.

Return Codes

`NF_NOERR` No error.

`NF_EBADTYPEID`
Bad typeid.

`NF_ENOTNC4`
Seeking a user-defined type in a netCDF-3 file.

`NF_ESTRICTNC3`
Seeking a user-defined type in a netCDF-4 file for which classic model has been turned on.

`NF_EBADGRPID`
Bad group ID in ncid.

`NF_EBADID`
Type ID not found.

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from the test program nf_test/ftst_vars3.F, and it uses all the possible inquiry functions on an enum type.

```
C     Check the enum type.
      retval = NF_INQ_TYPEIDS(ncid, num_types, typeids)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (num_types .ne. MAX_TYPES) stop 2
      retval = nf_inq_enum(ncid, typeids(1), type_name, base_type,
     &     base_size, num_members)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

```
      if (base_type .ne. NF_INT .or. num_members .ne. 2) stop 2
      retval = nf_inq_enum_member(ncid, typeids(1), 1, member_name,
     &     member_value)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (member_name(1:len(one_name)) .ne. one_name) stop 2
```

5.5 Learn About a User Defined Type: NF_INQ_USER_TYPE

Given an ncid and a typeid, get the information about a user defined type. This function will work on any user defined type, whether compound, opaque, enumeration, or variable length array.

Usage

```
INTEGER FUNCTION NF_INQ_USER_TYPE(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) NAME, INTEGER SIZE, INTEGER BASE_NF_TYPE,
        INTEGER NFIELDS, INTEGER CLASS)
```

`NCID` The ncid for the group containing the user defined type.

`XTYPE` The typeid for this type, as returned by NF_DEF_COMPOUND, NF_DEF_OPAQUE, NF_DEF_ENUM, NF_DEF_VLEN, or NF_INQ_VAR.

`NAME` The name of the user defined type will be copied here. It will be NF_MAX_NAME bytes or less.

`SIZE` The (in-memory) size of the user defined type will be copied here.

`BASE_NF_TYPE`
The base typeid will be copied here for vlen and enum types.

`NFIELDS` The number of fields will be copied here for enum and compound types.

`CLASS` The class of the user defined type, NF_VLEN, NF_OPAQUE, NF_ENUM, or NF_COMPOUND, will be copied here.

Errors

`NF_NOERR` No error.

`NF_EBADTYPEID`
Bad typeid.

`NF_EBADFIELDID`
Bad fieldid.

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_types2.F.

```
C     Check the type.
      retval = nf_inq_user_type(ncid, typeids(1), name_in, size_in,
     &     base_type_in, nfields_in, class_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.6 Compound Types Introduction

NetCDF-4 added support for compound types, which allow users to construct a new type - a combination of other types, like a C struct.

Compound types are not supported in classic or 64-bit offset format files.

To write data in a compound type, first use nf_def_compound to create the type, multiple calls to nf_insert_compound to add to the compound type, and then write data with the appropriate nf_put_var1, nf_put_vara, nf_put_vars, or nf_put_varm call.

To read data written in a compound type, you must know its structure. Use the NF_INQ_COMPOUND functions to learn about the compound type.

In Fortran a character buffer must be used for the compound data. The user must read the data from within that buffer in the same way that the C compiler which compiled netCDF would store the structure.

The use of compound types introduces challenges and portability issues for Fortran users.

5.6.1 Creating a Compound Type: NF_DEF_COMPOUND

Create a compound type. Provide an ncid, a name, and a total size (in bytes) of one element of the completed compound type.

After calling this function, fill out the type with repeated calls to NF_INSERT_COMPOUND (see Section 5.6.2 [NF_INSERT_COMPOUND], page 50). Call NF_INSERT_COMPOUND once for each field you wish to insert into the compound type.

Note that there does not seem to be a way to read such types into structures in Fortran 90 (and there are no structures in Fortran 77).

Fortran users may use character buffers to read and write compound types.

Usage

```
INTEGER FUNCTION NF_DEF_COMPOUND(INTEGER NCID, INTEGER SIZE,
          CHARACTER*(*) NAME, INTEGER TYPEIDP)
```

`NCID` The groupid where this compound type will be created.

`SIZE` The size, in bytes, of the compound type.

`NAME` The name of the new compound type.

`TYPEIDP` The typeid of the new type will be placed here.

Errors

NF_NOERR No error.

NF_EBADID
Bad group id.

NF_ENAMEINUSE
That name is in use. Compound type names must be unique in the data file.

NF_EMAXNAME
Name exceeds max length NF_MAX_NAME.

NF_EBADNAME
Name contains illegal characters.

NF_ENOTNC4
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag NF_NETCDF4. (see Section 2.8 [NF_OPEN], page 13).

NF_ESTRICTNC3
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

NF_EHDFERR
An error was reported by the HDF5 layer.

NF_EPERM Attempt to write to a read-only file.

NF_ENOTINDEFINE
Not in define mode.

Example

This example is from nf_test/ftst_types2.F.

```
C     Define a compound type.
      retval = nf_def_compound(ncid, cmp_size, type_name,
     &     cmp_typeid)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.6.2 Inserting a Field into a Compound Type: NF_INSERT_COMPOUND

Insert a named field into a compound type.

Usage

```
INTEGER FUNTION NF_INSERT_COMPOUND(INTEGER TYPEID, CHARACTER*(*) NAME, INTEGER OFFSET,
        INTEGER FIELD_TYPEID)
```

TYPEID The typeid for this compound type, as returned by NF_DEF_COMPOUND, or NF_INQ_VAR.

NAME The name of the new field.

`OFFSET` Offset in byte from the beginning of the compound type for this field.

`FIELD_TYPEID`
The type of the field to be inserted.

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENAMEINUSE`
That name is in use. Field names must be unique within a compound type.

`NF_EMAXNAME`
Name exceed max length NF_MAX_NAME.

`NF_EBADNAME`
Name contains illegal characters.

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag NF_NETCDF4. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
An error was reported by the HDF5 layer.

`NF_ENOTINDEFINE`
Not in define mode.

Example

This example is from nf_test/ftst_types.F.

```
C     Define a compound type.
      retval = nf_def_compound(ncid, WIND_T_SIZE, type_name,
     &     wind_typeid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_insert_compound(ncid, wind_typeid, u_name, 0, NF_INT)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_insert_compound(ncid, wind_typeid, v_name, 4, NF_INT)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.6.3 Inserting an Array Field into a Compound Type: NF_INSERT_ARRAY_COMPOUND

Insert a named array field into a compound type.

Usage

```
INTEGER FUNCTION NF_INSERT_ARRAY_COMPOUND(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) NAME, INTEGER OFFSET, INTEGER FIELD_TYPEID,
        INTEGER NDIMS, INTEGER DIM_SIZES)
```

`NCID` The ID of the file that contains the array type and the compound type.

`XTYPE` The typeid for this compound type, as returned by nf_def_compound, or nf_inq_var.

`NAME` The name of the new field.

`OFFSET` Offset in byte from the beginning of the compound type for this field.

`FIELD_TYPEID`
The base type of the array to be inserted.

`NDIMS` The number of dimensions for the array to be inserted.

`DIM_SIZES`
An array containing the sizes of each dimension.

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENAMEINUSE`
That name is in use. Field names must be unique within a compound type.

`NF_EMAXNAME`
Name exceed max length NF_MAX_NAME.

`NF_EBADNAME`
Name contains illegal characters.

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag NF_NETCDF4. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
An error was reported by the HDF5 layer.

`NF_ENOTINDEFINE`
Not in define mode.

`NF_ETYPEDEFINED`
Attempt to change type that has already been committed. The first time the file leaves define mode, all defined types are committed, and can't be changed.

If you wish to add an array to a compound type, you must do so before the compound type is committed.

Example

This example is from nf_test/ftst_types2.F.

```
C     Define a compound type.
      retval = nf_def_compound(ncid, cmp_size, type_name,
     &     cmp_typeid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Include an array.
      dim_sizes(1) = NX
      dim_sizes(2) = NY
      retval = nf_insert_array_compound(ncid, cmp_typeid, ary_name, 0,
     &     NF_INT, NDIMS, dim_sizes)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.6.4 Learn About a Compound Type: NF_INQ_COMPOUND

Get the number of fields, length in bytes, and name of a compound type.

In addtion to the NF_INQ_COMPOUND function, three additional functions are provided which get only the name, size, and number of fields.

Usage

```
INTEGER FUNCTION NF_INQ_COMPOUND(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) NAME, INTEGER SIZEP, INTEGER NFIELDSP)

INTEGER FUNCTION NF_INQ_COMPOUND_NAME(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) NAME)

INTEGER FUNCTION NF_INQ_COMPOUND_SIZE(INTEGER NCID, INTEGER XTYPE,
        INTEGER SIZEP)

INTEGER FUNCTION NF_INQ_COMPOUND_NFIELDS(INTEGER NCID, INTEGER XTYPE,
        INTEGER NFIELDSP)
```

`NCID` The ID of any group in the file that contains the compound type.

`XTYPE` The typeid for this compound type, as returned by NF_DEF_COMPOUND, or NF_INQ_VAR.

`NAME` Character array which will get the name of the compound type. It will have a maximum length of NF_MAX_NAME.

`SIZEP` The size of the compound type in bytes will be put here.

`NFIELDSP` The number of fields in the compound type will be placed here.

Return Codes

`NF_NOERR` No error.

`NF_EBADID`
: Couldn't find this ncid.

`NF_ENOTNC4`
: Not a netCDF-4/HDF5 file.

`NF_ESTRICTNC3`
: A netCDF-4/HDF5 file, but with CLASSIC_MODEL. No user defined types are allowed in the classic model.

`NF_EBADTYPE`
: This type not a compound type.

`NF_EBADTYPEID`
: Bad type id.

`NF_EHDFERR`
: An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_types.F.

```
C     Check it differently.
      retval = nf_inq_compound(ncid, typeids(1), name_in, size_in,
     &     nfields_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (name_in(1:len(type_name)) .ne. type_name .or.
     &     size_in .ne. WIND_T_SIZE .or. nfields_in .ne. 2) stop 2

C     Check it one piece at a time.
      retval = nf_inq_compound_nfields(ncid, typeids(1), nfields_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (nfields_in .ne. 2) stop 2
      retval = nf_inq_compound_size(ncid, typeids(1), size_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (size_in .ne. WIND_T_SIZE) stop 2
      retval = nf_inq_compound_name(ncid, typeids(1), name_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (name_in(1:len(type_name)) .ne. type_name) stop 2
```

5.6.5 Learn About a Field of a Compound Type: NF_INQ_COMPOUND_FIELD

Get information about one of the fields of a compound type.

Usage

```
INTEGER FUNCTION NF_INQ_COMPOUND_FIELD(INTEGER NCID, INTEGER XTYPE,
```

```
        INTEGER FIELDID, CHARACTER*(*) NAME, INTEGER OFFSETP,
        INTEGER FIELD_TYPEIDP, INTEGER NDIMSP, INTEGER DIM_SIZESP)

INTEGER FUNCTION NF_INQ_COMPOUND_FIELDNAME(INTEGER TYPEID,
        INTEGER FIELDID, CHARACTER*(*) NAME)

INTEGER FUNCTION NF_INQ_COMPOUND_FIELDINDEX(INTEGER TYPEID,
        CHARACTER*(*) NAME, INTEGER FIELDIDP)

INTEGER FUNCTION NF_INQ_COMPOUND_FIELDOFFSET(INTEGER TYPEID,
        INTEGER FIELDID, INTEGER OFFSETP)

INTEGER FUNCTION NF_INQ_COMPOUND_FIELDTYPE(INTEGER TYPEID,
        INTEGER FIELDID, INTEGER FIELD_TYPEIDP)

INTEGER FUNCTION NF_INQ_COMPOUND_FIELDNDIMS(INTEGER NCID,
        INTEGER XTYPE, INTEGER FIELDID, INTEGER NDIMSP)

INTEGER FUNCTION NF_INQ_COMPOUND_FIELDDIM_SIZES(INTEGER NCID,
        INTEGER XTYPE, INTEGER FIELDID, INTEGER DIM_SIZES)
```

`NCID` The groupid where this compound type exists.

`XTYPE` The typeid for this compound type, as returned by NF_DEF_COMPOUND, or NF_INQ_VAR.

`FIELDID` A one-based index number specifying a field in the compound type.

`NAME` A character array which will get the name of the field. The name will be NF_MAX_NAME characters, at most.

`OFFSETP` An integer which will get the offset of the field.

`FIELD_TYPEID`
An integer which will get the typeid of the field.

`NDIMSP` An integer which will get the number of dimensions of the field.

`DIM_SIZESP`
An integer array which will get the dimension sizes of the field.

Errors

`NF_NOERR` No error.

`NF_EBADTYPEID`
Bad type id.

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/fst_types.F.

```
C     Check the first field of the compound type.
      retval = nf_inq_compound_field(ncid, typeids(1), 1, name_in,
     &     offset_in, field_typeid_in, ndims_in, dim_sizes_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (name_in(1:len(u_name)) .ne. u_name .or. offset_in .ne. 0 .or.
     &     field_typeid_in .ne. NF_INT .or. ndims_in .ne. 0) stop 2
      retval = nf_inq_compound_fieldname(ncid, typeids(1), 1, name_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (name_in(1:len(u_name)) .ne. u_name) stop 2
      retval = nf_inq_compound_fieldoffset(ncid, typeids(1), 1,
     &     offset_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (offset_in .ne. 0) stop 2
      retval = nf_inq_compound_fieldtype(ncid, typeids(1), 1,
     &     field_typeid_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (field_typeid_in .ne. NF_INT) stop 2
      retval = nf_inq_compound_fieldndims(ncid, typeids(1), 1,
     &     ndims_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (ndims_in .ne. 0) stop 2
```

5.7 Variable Length Array Introduction

NetCDF-4 added support for a variable length array type. This is not supported in classic or 64-bit offset files, or in netCDF-4 files which were created with the NF_CLASSIC_MODEL flag.

A variable length array is represented in C as a structure from HDF5, the nf_vlen_t structure. It contains a len member, which contains the length of that array, and a pointer to the array.

So an array of VLEN in C is an array of nc_vlen_t structures. The only way to handle this in Fortran is with a character buffer sized correctly for the platform.

The extra access functions NF_GET_VLEN_ELEMENT and NF_PUT_VLEN_ELEMENT to get and put one VLEN element. (That is, one array of variable length.) When calling the put, the data are not copied from the source. When calling the get the data are copied from VLEN allocated memory, which must still be freed (see below).

VLEN arrays are handled differently with respect to allocation of memory. Generally, when reading data, it is up to the user to malloc (and subsequently free) the memory needed to hold the data. It is up to the user to ensure that enough memory is allocated.

With VLENs, this is impossible. The user cannot know the size of an array of VLEN until after reading the array. Therefore when reading VLEN arrays, the netCDF library will allocate the memory for the data within each VLEN.

It is up to the user, however, to eventually free this memory. This is not just a matter of one call to free, with the pointer to the array of VLENs; each VLEN contains a pointer which must be freed.

Compression is permitted but may not be effective for VLEN data, because the compression is applied to the nc_vlen_t structures, rather than the actual data.

5.7.1 Define a Variable Length Array (VLEN): NF_DEF_VLEN

Use this function to define a variable length array type.

Usage

```
INTEGER FUNCTION NF_DEF_VLEN(INTEGER NCID, CHARACTER*(*) NAME,
        INTEGER BASE_TYPEID, INTEGER XTYPEP)
```

`NCID` The ncid of the file to create the VLEN type in.

`NAME` A name for the VLEN type.

`BASE_TYPEID`
: The typeid of the base type of the VLEN. For example, for a VLEN of shorts, the base type is NF_SHORT. This can be a user defined type.

`XTYPEP` The typeid of the new VLEN type will be set here.

Errors

`NF_NOERR` No error.

`NF_EMAXNAME`
: NF_MAX_NAME exceeded.

`NF_ENAMEINUSE`
: Name is already in use.

`NF_EBADNAME`
: Attribute or variable name contains illegal characters.

`NF_EBADID`
: ncid invalid.

`NF_EBADGRPID`
: Group ID part of ncid was invalid.

`NF_EINVAL`
: Size is invalid.

`NF_ENOMEM`
: Out of memory.

Example

This example is from nf_test/ftst_vars4.F.

```
C     Create the vlen type.
      retval = nf_def_vlen(ncid, vlen_type_name, nf_int, vlen_typeid)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.7.2 Learning about a Variable Length Array (VLEN) Type: NF_INQ_VLEN

Use this type to learn about a vlen.

Usage

```
INTEGER FUNCTION NF_INQ_VLEN(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) NAME, INTEGER DATUM_SIZEP, INTEGER
        BASE_NF_TYPEP)
```

`NCID` The ncid of the file that contains the VLEN type.

`XTYPE` The type of the VLEN to inquire about.

`NAME` The name of the VLEN type. The name will be NF_MAX_NAME characters or less.

`DATUM_SIZEP`
A pointer to a size_t, this will get the size of one element of this vlen.

`BASE_NF_TYPEP`
An integer that will get the type of the VLEN base type. (In other words, what type is this a VLEN of?)

Errors

`NF_NOERR` No error.

`NF_EBADTYPE`
Can't find the typeid.

`NF_EBADID`
ncid invalid.

`NF_EBADGRPID`
Group ID part of ncid was invalid.

Example

This example is from nf_test/ftst_vars4.F.

```
C     Use nf_inq_vlen and make sure we get the same answers as we did
C     with nf_inq_user_type.
      retval = nf_inq_vlen(ncid, typeids(1), type_name, base_size,
     &     base_type)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.7.3 Releasing Memory for a Variable Length Array (VLEN) Type: NF_FREE_VLEN

When a VLEN is read into user memory from the file, the HDF5 library performs memory allocations for each of the variable length arrays contained within the VLEN structure. This memory must be freed by the user to avoid memory leaks.

This violates the normal netCDF expectation that the user is responsible for all memory allocation. But, with VLEN arrays, the underlying HDF5 library allocates the memory for the user, and the user is responsible for deallocating that memory.

Usage

```
INTEGER FUNCTION NF_FREE_VLEN(CHARACTER VL);
```

`VL` The variable length array structure which is to be freed.

Errors

`NF_NOERR` No error.

`NF_EBADTYPE`
Can't find the typeid.

Example

5.7.4 Set a Variable Length Array with NF_PUT_VLEN_ELEMENT

Use this to set the element of the (potentially) n-dimensional array of VLEN. That is, this sets the data in one variable length array.

Usage

```
INTEGER FUNCTION NF_PUT_VLEN_ELEMENT(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) VLEN_ELEMENT, INTEGER LEN, DATA)
```

`NCID` The ncid of the file that contains the VLEN type.

`XTYPE` The type of the VLEN.

`VLEN_ELEMENT`
The VLEN element to be set.

`LEN` The number of entries in this array.

`DATA` The data to be stored. Must match the base type of this VLEN.

Errors

`NF_NOERR` No error.

`NF_EBADTYPE`
Can't find the typeid.

`NF_EBADID`
ncid invalid.

`NF_EBADGRPID`
Group ID part of ncid was invalid.

Example

This example is from nf_test/ftst_vars4.F.

```
C     Set up the vlen with this helper function, since F77 can't deal
C     with pointers.
      retval = nf_put_vlen_element(ncid, vlen_typeid, vlen,
     &     vlen_len, data1)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.7.5 Set a Variable Length Array with NF_GET_VLEN_ELEMENT

Use this to set the element of the (potentially) n-dimensional array of VLEN. That is, this sets the data in one variable length array.

Usage

```
INTEGER FUNCTION NF_GET_VLEN_ELEMENT(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) VLEN_ELEMENT, INTEGER LEN, DATA)
```

`NCID` The ncid of the file that contains the VLEN type.

`XTYPE` The type of the VLEN.

`VLEN_ELEMENT`
: The VLEN element to be set.

`LEN` This will be set to the number of entries in this array.

`DATA` The data will be copied here. Sufficient storage must be available or bad things will happen to you.

Errors

`NF_NOERR` No error.

`NF_EBADTYPE`
: Can't find the typeid.

`NF_EBADID`
: ncid invalid.

`NF_EBADGRPID`
: Group ID part of ncid was invalid.

Example

This example is from nf_test/ftst_vars4.F.

```
C     Read the vlen attribute.
      retval = nf_get_att(ncid, NF_GLOBAL, 'att1', vlen_in)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Get the data from the vlen we just read.
```

```
          retval = nf_get_vlen_element(ncid, vlen_typeid, vlen_in,
         &     vlen_len_in, data1_in)
          if (retval .ne. nf_noerr) call handle_err(retval)
```

5.8 Opaque Type Introduction

NetCDF-4 added support for the opaque type. This is not supported in classic or 64-bit offset files.

The opaque type is a type which is a collection of objects of a known size. (And each object is the same size). Nothing is known to netCDF about the contents of these blobs of data, except their size in bytes, and the name of the type.

To use an opaque type, first define it with Section 5.8.1 [NF_DEF_OPAQUE], page 61. If encountering an enum type in a new data file, use Section 5.8.2 [NF_INQ_OPAQUE], page 62 to learn its name and size.

5.8.1 Creating Opaque Types: NF_DEF_OPAQUE

Create an opaque type. Provide a size and a name.

Usage

```
INTEGER FUNCTION NF_DEF_OPAQUE(INTEGER NCID, INTEGER SIZE,
        CHARACTER*(*) NAME, INTEGER TYPEIDP)
```

`NCID` The groupid where the type will be created. The type may be used anywhere in the file, no matter what group it is in.

`SIZE` The size of each opaque object.

`NAME` The name for this type. Must be shorter than NF_MAX_NAME.

`TYPEIDP` Pointer where the new typeid for this type is returned. Use this typeid when defining variables of this type with Section 6.3 [NF_DEF_VAR], page 70.

Errors

`NF_NOERR` No error.

`NF_EBADTYPEID`
Bad typeid.

`NF_EBADFIELDID`
Bad fieldid.

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_vars3.F.

```
C     Create the opaque type.
      retval = nf_def_opaque(ncid, opaque_size, opaque_type_name,
     &     opaque_typeid)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.8.2 Learn About an Opaque Type: NF_INQ_OPAQUE

Given a typeid, get the information about an opaque type.

Usage

```
INTEGER FUNCTION NF_INQ_OPAQUE(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) NAME, INTEGER SIZEP)
```

`NCID` The ncid for the group containing the opaque type.

`XTYPE` The typeid for this opaque type, as returned by NF_DEF_COMPOUND, or NF_INQ_VAR.

`NAME` The name of the opaque type will be copied here. It will be NF_MAX_NAME bytes or less.

`SIZEP` The size of the opaque type will be copied here.

Errors

`NF_NOERR` No error.

`NF_EBADTYPEID`
Bad typeid.

`NF_EBADFIELDID`
Bad fieldid.

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_vars3.F.

```
C     Use nf_inq_opaque and make sure we get the same answers as we did
C     with nf_inq_user_type.
      retval = nf_inq_opaque(ncid, typeids(2), type_name, base_size)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.9 Enum Type Introduction

NetCDF-4 added support for the enum type. This is not supported in classic or 64-bit offset files.

5.9.1 Creating a Enum Type: NF_DEF_ENUM

Create an enum type. Provide an ncid, a name, and a base integer type.

After calling this function, fill out the type with repeated calls to NF_INSERT_ENUM (see Section 5.9.2 [NF_INSERT_ENUM], page 63). Call NF_INSERT_ENUM once for each value you wish to make part of the enumeration.

Usage

```
INTEGER FUNCTION NF_DEF_ENUM(INTEGER NCID, INTEGER BASE_TYPEID,
        CHARACTER*(*) NAME, INTEGER TYPEIDP)
```

`NCID` The groupid where this compound type will be created.

`BASE_TYPEID`
: The base integer type for this enum. Must be one of: NF_BYTE, NF_UBYTE, NF_SHORT, NF_USHORT, NF_INT, NF_UINT, NF_INT64, NF_UINT64.

`NAME` The name of the new enum type.

`TYPEIDP` The typeid of the new type will be placed here.

Errors

`NF_NOERR` No error.

`NF_EBADID`
: Bad group id.

`NF_ENAMEINUSE`
: That name is in use. Compound type names must be unique in the data file.

`NF_EMAXNAME`
: Name exceeds max length NF_MAX_NAME.

`NF_EBADNAME`
: Name contains illegal characters.

`NF_ENOTNC4`
: Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag NF_NETCDF4. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
: This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
: An error was reported by the HDF5 layer.

`NF_EPERM` Attempt to write to a read-only file.

`NF_ENOTINDEFINE`
: Not in define mode.

This example is from nf_test/ftst_vars3.F.

```
C     Create the enum type.
      retval = nf_def_enum(ncid, NF_INT, enum_type_name, enum_typeid)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.9.2 Inserting a Field into a Enum Type: NF_INSERT_ENUM

Insert a named member into a enum type.

Usage

```
INTEGER FUNCTION NF_INSERT_ENUM(INTEGER NCID, INTEGER XTYPE,
        CHARACTER IDENTIFIER, INTEGER VALUE)
```

`NCID` The ncid of the group which contains the type.

`TYPEID` The typeid for this enum type, as returned by nf_def_enum, or nf_inq_var.

`IDENTIFIER`
The identifier of the new member.

`VALUE` The value that is to be associated with this member.

Errors

`NF_NOERR` No error.

`NF_EBADID`
Bad group id.

`NF_ENAMEINUSE`
That name is in use. Field names must be unique within a enum type.

`NF_EMAXNAME`
Name exceed max length NF_MAX_NAME.

`NF_EBADNAME`
Name contains illegal characters.

`NF_ENOTNC4`
Attempting a netCDF-4 operation on a netCDF-3 file. NetCDF-4 operations can only be performed on files defined with a create mode which includes flag NF_NETCDF4. (see Section 2.8 [NF_OPEN], page 13).

`NF_ESTRICTNC3`
This file was created with the strict netcdf-3 flag, therefore netcdf-4 operations are not allowed. (see Section 2.8 [NF_OPEN], page 13).

`NF_EHDFERR`
An error was reported by the HDF5 layer.

`NF_ENOTINDEFINE`
Not in define mode.

Example

This example is from nf_test/ftst_vars3.F.

```
      one = 1
      zero = 0
      retval = nf_insert_enum(ncid, enum_typeid, zero_name, zero)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_insert_enum(ncid, enum_typeid, one_name, one)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

5.9.3 Learn About a Enum Type: NF_INQ_ENUM

Get information about a user-defined enumeration type.

Usage

```
INTEGER FUNCTION NF_INQ_ENUM(INTEGER NCID, INTEGER XTYPE,
        CHARACTER*(*) NAME, INTEGER BASE_NF_TYPE, INTEGER BASE_SIZE,
        INTEGER NUM_MEMBERS)
```

`NCID` The group ID of the group which holds the enum type.

`XTYPE` The typeid for this enum type, as returned by NF_DEF_ENUM, or NF_INQ_VAR.

`NAME` Character array which will get the name. It will have a maximum length of NF_MAX_NAME.

`BASE_NF_TYPE`
An integer which will get the base integer type of this enum.

`BASE_SIZE`
An integer which will get the size (in bytes) of the base integer type of this enum.

`NUM_MEMBERS`
An integer which will get the number of members defined for this enumeration type.

Errors

`NF_NOERR` No error.

`NF_EBADTYPEID`
Bad type id.

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

In this example from nf_test/ftst_vars3.F, an enum type is created and then examined:

```
 retval = nf_inq_enum(ncid, typeids(1), type_name, base_type,
&     base_size, num_members)
 if (retval .ne. nf_noerr) call handle_err(retval)
 if (base_type .ne. NF_INT .or. num_members .ne. 2) stop 2
```

5.9.4 Learn the Name of a Enum Type: nf_inq_enum_member

Get information about a member of an enum type.

Usage

```
INTEGER FUNCTION NF_INQ_ENUM_MEMBER(INTEGER NCID, INTEGER XTYPE,
        INTEGER IDX, CHARACTER*(*) NAME, INTEGER VALUE)
```

`NCID` The groupid where this enum type exists.

`XTYPE` The typeid for this enum type.

`IDX` The one-based index number for the member of interest.

`NAME` A character array which will get the name of the member. It will have a maximum length of NF_MAX_NAME.

`VALUE` An integer that will get the value associated with this member.

Errors

`NF_NOERR` No error.

`NF_EBADTYPEID`
Bad type id.

`NF_EHDFERR`
An error was reported by the HDF5 layer.

Example

This example is from nf_test/ftst_vars3.F:

```
C     Check the members of the enum type.
      retval = nf_inq_enum_member(ncid, typeids(1), 1, member_name,
     &     member_value)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (member_name(1:len(zero_name)) .ne. zero_name .or.
     &     member_value .ne. 0) stop 2
      retval = nf_inq_enum_member(ncid, typeids(1), 2, member_name,
     &     member_value)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (member_name(1:len(one_name)) .ne. one_name .or.
     &     member_value .ne. 1) stop 2
```

5.9.5 Learn the Name of a Enum Type: NF_INQ_ENUM_IDENT

Get the name which is associated with an enum member value.

This is similar to NF_INQ_ENUM_MEMBER, but instead of using the index of the member, you use the value of the member.

Usage

```
INTEGER FUNCTION NF_INQ_ENUM_IDENT(INTEGER NCID, INTEGER XTYPE,
        INTEGER VALUE, CHARACTER*(*) IDENTIFIER)
```

`NCID` The groupid where this enum type exists.

`XTYPE` The typeid for this enum type.

`VALUE` The value for which an identifier is sought.

`IDENTIFIER`
A character array that will get the identifier. It will have a maximum length of NF_MAX_NAME.

Return Code

`NF_NOERR` No error.

`NF_EBADTYPEID`
Bad type id, or not an enum type.

`NF_EHDFERR`
An error was reported by the HDF5 layer.

`NF_EINVAL`
The value was not found in the enum.

Example

In this example from nf_test/ftst_vars3.F, the values for 0 and 1 are checked in an enum.

```
retval = nf_inq_enum_ident(ncid, typeids(1), 0, member_name)
if (retval .ne. nf_noerr) call handle_err(retval)
if (member_name(1:len(zero_name)) .ne. zero_name) stop 2
retval = nf_inq_enum_ident(ncid, typeids(1), 1, member_name)
if (retval .ne. nf_noerr) call handle_err(retval)
if (member_name(1:len(one_name)) .ne. one_name) stop 2
```

6 Variables

6.1 Variables Introduction

Variables for a netCDF dataset are defined when the dataset is created, while the netCDF dataset is in define mode. Other variables may be added later by reentering define mode. A netCDF variable has a name, a type, and a shape, which are specified when it is defined. A variable may also have values, which are established later in data mode.

Ordinarily, the name, type, and shape are fixed when the variable is first defined. The name may be changed, but the type and shape of a variable cannot be changed. However, a variable defined in terms of the unlimited dimension can grow without bound in that dimension.

A netCDF variable in an open netCDF dataset is referred to by a small integer called a variable ID.

Variable IDs reflect the order in which variables were defined within a netCDF dataset. Variable IDs are 1, 2, 3,..., in the order in which the variables were defined. A function is available for getting the variable ID from the variable name and vice-versa.

Attributes (see Chapter 7 [Attributes], page 119) may be associated with a variable to specify such properties as units.

Operations supported on variables are:

- Create a variable, given its name, data type, and shape.
- Get a variable ID from its name.
- Get a variable's name, data type, shape, and number of attributes from its ID.
- Put a data value into a variable, given variable ID, indices, and value.
- Put an array of values into a variable, given variable ID, corner indices, edge lengths, and a block of values.
- Put a subsampled or mapped array-section of values into a variable, given variable ID, corner indices, edge lengths, stride vector, index mapping vector, and a block of values.
- Get a data value from a variable, given variable ID and indices.
- Get an array of values from a variable, given variable ID, corner indices, and edge lengths.
- Get a subsampled or mapped array-section of values from a variable, given variable ID, corner indices, edge lengths, stride vector, and index mapping vector.
- Rename a variable.

6.2 Language Types Corresponding to netCDF external data types

The following table gives the netCDF external data types and the corresponding type constants for defining variables in the FORTRAN interface:

Type	FORTRAN API Mnemonic	Bits
byte	NF_BYTE	8

char	NF_CHAR	8
short	NF_SHORT	16
int	NF_INT	32
float	NF_FLOAT	32
double	NF_DOUBLE	64

The first column gives the netCDF external data type, which is the same as the CDL data type. The next column gives the corresponding FORTRAN parameter for use in netCDF functions (the parameters are defined in the netCDF FORTRAN include-file netcdf.inc). The last column gives the number of bits used in the external representation of values of the corresponding type.

Note that there are no netCDF types corresponding to 64-bit integers or to characters wider than 8 bits in the current version of the netCDF library.

6.3 Create a Variable: `NF_DEF_VAR`

The function NF_DEF_VAR adds a new variable to an open netCDF dataset in define mode. It returns (as an argument) a variable ID, given the netCDF ID, the variable name, the variable type, the number of dimensions, and a list of the dimension IDs.

Usage

```
INTEGER FUNCTION NF_DEF_VAR(INTEGER NCID, CHARACTER*(*) NAME,
                            INTEGER XTYPE, INTEGER NVDIMS,
                            INTEGER VDIMS(*), INTEGER varid)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`NAME` Variable name.

`XTYPE` One of the set of predefined netCDF external data types. The type of this parameter, NF_TYPE, is defined in the netCDF header file. The valid netCDF external data types are NF_BYTE, NF_CHAR, NF_SHORT, NF_INT, NF_FLOAT, and NF_DOUBLE. If the file is a NetCDF-4/HDF5 file, the additional types NF_UBYTE, NF_USHORT, NF_UINT, NF_INT64, NF_UINT64, and NF_STRING may be used, as well as a user defined type ID.

`NVDIMS` Number of dimensions for the variable. For example, 2 specifies a matrix, 1 specifies a vector, and 0 means the variable is a scalar with no dimensions. Must not be negative or greater than the predefined constant NF_MAX_VAR_DIMS.

`VDIMS` Vector of ndims dimension IDs corresponding to the variable dimensions. If the ID of the unlimited dimension is included, it must be first. This argument is ignored if ndims is 0. For expanded model netCDF4/HDF5 files, there may be any number of unlimited dimensions, and they may be used in any element of the dimids array.

`varid` Returned variable ID.

Errors

NF_DEF_VAR returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The netCDF dataset is not in define mode.
- The specified variable name is the name of another existing variable.
- The specified type is not a valid netCDF type.
- The specified number of dimensions is negative or more than the constant NF_MAX_VAR_DIMS, the maximum number of dimensions permitted for a netCDF variable.
- One or more of the dimension IDs in the list of dimensions is not a valid dimension ID for the netCDF dataset.
- The number of variables would exceed the constant NF_MAX_VARS, the maximum number of variables permitted in a netCDF dataset.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_DEF_VAR to create a variable named rh of type double with three dimensions, time, lat, and lon in a new netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  STATUS, NCID
INTEGER  LATDIM, LONDIM, TIMDIM  ! dimension IDs
INTEGER  RHID                    ! variable ID
INTEGER  RHDIMS(3)               ! variable shape
   ...
STATUS = NF_CREATE ('foo.nc', NF_NOCLOBBER, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
                                 ! define dimensions
STATUS = NF_DEF_DIM(NCID, 'lat', 5, LATDIM)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_DEF_DIM(NCID, 'lon', 10, LONDIM)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_DEF_DIM(NCID, 'time', NF_UNLIMITED, TIMDIM)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
         ...
                                 ! define variable
RHDIMS(1) = LONDIM
RHDIMS(2) = LATDIM
RHDIMS(3) = TIMDIM
STATUS = NF_DEF_VAR (NCID, 'rh', NF_DOUBLE, 3, RHDIMS, RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.4 Define Chunking Parameters for a Variable: NF_DEF_VAR_CHUNKING

The function NF_DEF_VAR_CHUNKING sets the storage parameters for a variable in a netCDF-4 file. It can set the chunk sizes to get chunked storage, or it can set the contiguous flag to get contiguous storage.

Variables that make use of one or more unlimited dimensions, compression, or checksums must use chunking. Such variables are created with default chunk sizes of 1 for each unlimited dimension and the dimension length for other dimensions, except that if the resulting chunks are too large, the default chunk sizes for non-record dimensions are reduced.

The total size of a chunk must be less than 4 GiB. That is, the product of all chunksizes and the size of the data (or the size of nc_vlen_t for VLEN types) must be less than 4 GiB.

This function may only be called after the variable is defined, but before nc_enddef is called. Once the chunking parameters are set for a variable, they cannot be changed. This function can be used to change the default chunking for record, compressed, or checksummed variables before nc_enddef is called.

Note that you cannot set chunking for scalar variables. Only non-scalar variables can have chunking.

Usage

```
NF_DEF_VAR_CHUNKING(INTEGER NCID, INTEGER VARID, INTEGER STORAGE, INTEGER CHUNKSIZES)
```

`ncid` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`varid` Variable ID.

`storage` If NF_CONTIGUOUS, then contiguous storage is used for this variable. Variables with compression, shuffle filter, checksums, or one or more unlimited dimensions cannot use contiguous storage. If contiguous storage is turned on, the chunksizes parameter is ignored.

If NF_CHUNKED, then chunked storage is used for this variable. Chunk sizes may be specified with the chunksizes parameter. Default sizes will be used if chunking is required and this function is not called.

By default contiguous storage is used for fix-sized variables when conpression, chunking, checksums, or endianness control are not used.

`chunksizes`
An array of chunk sizes. The array must have the one chunksize for each dimension in the variable. If contiguous storage is used, then the chunksizes parameter is ignored.

Errors

NF_DEF_VAR_CHUNKING returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_EINVAL`
: Invalid input. This can occur when the user attempts to set contiguous storage for a variable with compression or checksums, or one or more unlimited dimensions.

`NF_ENOTNC4`
: Not a netCDF-4 file.

`NF_ENOTVAR`
: Can't find this variable.

`NF_ELATEDEF`
: This variable has already been the subject of a NF_ENDDEF call. In netCDF-4 files NF_ENDDEF will be called automatically for any data read or write. Once enddef has been called, it is impossible to set the chunking for a variable.

`NF_ENOTINDEFINE`
: Not in define mode. This is returned for netCDF classic or 64-bit offset files, or for netCDF-4 files, when they were been created with NF_STRICT_NC3 flag. (see Section 2.5 [NF_CREATE], page 9).

`NF_ESTRICTNC3`
: Trying to create a var some place other than the root group in a netCDF file with NF_STRICT_NC3 turned on.

Example

In this example from nf_test/ftst_vars.F, a file is created, two dimensions and a variable are defined, and the chunksizes of the data are set to the size of the data (that is, data will be written in one chunk).

```
C     Create the netCDF file.
      retval = nf_create(FILE_NAME, NF_NETCDF4, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the dimensions.
      retval = nf_def_dim(ncid, "x", NX, x_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, "y", NY, y_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the variable.
      dimids(1) = y_dimid
      dimids(2) = x_dimid
      retval = NF_DEF_VAR(ncid, "data", NF_INT, NDIMS, dimids, varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Turn on chunking.
      chunks(1) = NY
      chunks(2) = NX
      retval = NF_DEF_VAR_chunking(ncid, varid, NF_CHUNKED, chunks)
```

```
        if (retval .ne. nf_noerr) call handle_err(retval)
```

6.5 Learn About Chunking Parameters for a Variable: NF_INQ_VAR_CHUNKING

The function NF_INQ_VAR_CHUNKING returns the chunking settings for a variable in a netCDF-4 file.

Usage

```
NF_INQ_VAR_CHUNKING(INTEGER NCID, INTEGER VARID, INTEGER STORAGE, INTEGER CHUNKSIZES);
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`STORAGE` On return, set to NF_CONTIGUOUS if this variable uses contiguous storage, NF_CHUNKED if it uses chunked storage.

`CHUNKSIZES`
An array of chunk sizes. The length of CHUNKSIZES must be the same as the number of dimensions of the variable.

Errors

NF_INQ_VAR_CHUNKING returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_ENOTVAR`
Can't find this variable.

Example

In this example from nf_test/ftst_vars.F, a variable with chunked storage is checked to ensure that the chunksizes are set to expected values.

```
C     Is everything set that is supposed to be?
      retval = nf_inq_var_chunking(ncid, varid, storage, chunks_in)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (storage .ne. NF_CHUNKED) stop 2
      if (chunks(1) .ne. chunks_in(1)) stop 2
      if (chunks(2) .ne. chunks_in(2)) stop 2
```

6.6 Set HDF5 Chunk Cache for a Variable: NF_SET_VAR_CHUNK_CACHE

This function changes the chunk cache settings for a variable. The change in cache size happens immediately. This is a property of the open file - it does not persist the next time you open the file.

For more information, see the documentation for the H5Pset_cache() function in the HDF5 library at the HDF5 website: `http://hdfgroup.org/HDF5/`.

Usage

```
nc_set_var_chunk_cache(int ncid, int varid, size_t size, size_t nelems,
       float preemption);
```

`ncid` NetCDF ID, from a previous call to nc_open or nc_create.

`varid` Variable ID.

`size` The total size of the raw data chunk cache, in megabytes. This should be big enough to hold multiple chunks of data. (Note that the C API uses bytes, but the Fortran APIs uses megabytes to avoid numbers that can't fit in 4-byte integers.)

`nelems` The number of chunk slots in the raw data chunk cache hash table. This should be a prime number larger than the number of chunks that will be in the cache.

`preemption`
The preemtion value must be between 0 and 100 inclusive and indicates how much chunks that have been fully read are favored for preemption. A value of zero means fully read chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 100 means fully read chunks are always preempted before other chunks. (The C API uses a float between 0 and 1 for this value).

Return Codes

`NF_NOERR` No error.

`NF_EINVAL`
Preemption must be between zero and 100 (inclusive).

Example

This example is from nf_test/ftst_vars2.F:

```
      include 'netcdf.inc'
 ...
C     These will be used to set the per-variable chunk cache.
      integer CACHE_SIZE, CACHE_NELEMS, CACHE_PREEMPTION
      parameter (CACHE_SIZE = 8, CACHE_NELEMS = 571)
      parameter (CACHE_PREEMPTION = 42)
...
C        Set variable caches.
```

```
            retval = nf_set_var_chunk_cache(ncid, varid(i), CACHE_SIZE,
        &        CACHE_NELEMS, CACHE_PREEMPTION)
            if (retval .ne. nf_noerr) call handle_err(retval)
```

6.7 Get the HDF5 Chunk Cache Settings for a variable: NF_GET_VAR_CHUNK_CACHE

This function gets the current chunk cache settings for a variable in a netCDF-4/HDF5 file.

For more information, see the documentation for the H5Pget_cache() function in the HDF5 library at the HDF5 website: `http://hdfgroup.org/HDF5/`.

Usage

```
INTEGER NF_GET_VAR_CHUNK_CACHE(INTEGER NCID, INTEGER VARID, INTEGER SIZE, INTEGER NELEMS,
                               INTEGER PREEMPTION);
```

`ncid` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`varid` Variable ID.

`sizep` The total size of the raw data chunk cache, in megabytes, will be put here.

`nelemsp` The number of chunk slots in the raw data chunk cache hash table will be put here.

`preemptionp`
The preemption will be put here. The preemtion value is between 0 and 100 inclusive and indicates how much chunks that have been fully read are favored for preemption. A value of zero means fully read chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 100 means fully read chunks are always preempted before other chunks.

Return Codes

`NC_NOERR` No error.

Example

This example is from nf_test/ftst_vars2.c:

```
        include 'netcdf.inc'
...
C       These will be used to set the per-variable chunk cache.
        integer CACHE_SIZE, CACHE_NELEMS, CACHE_PREEMPTION
        parameter (CACHE_SIZE = 8, CACHE_NELEMS = 571)
        parameter (CACHE_PREEMPTION = 42)

C       These will be used to check the setting of the per-variable chunk
C       cache.
        integer cache_size_in, cache_nelems_in, cache_preemption_in
```

```
...
        retval = nf_get_var_chunk_cache(ncid, varid(i), cache_size_in,
     &        cache_nelems_in, cache_preemption_in)
        if (retval .ne. nf_noerr) call handle_err(retval)
        if (cache_size_in .ne. CACHE_SIZE .or. cache_nelems_in .ne.
     &        CACHE_NELEMS .or. cache_preemption .ne. CACHE_PREEMPTION)
     &        stop 8
```

6.8 Define Fill Parameters for a Variable: `nf_def_var_fill`

The function NF_DEF_VAR_FILL sets the fill parameters for a variable in a netCDF-4 file.

This function must be called after the variable is defined, but before NF_ENDDEF is called.

Usage

```
NF_DEF_VAR_FILL(INTEGER NCID, INTEGER VARID, INTEGER NO_FILL, FILL_VALUE);
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`NO_FILL` Set to non-zero value to set no_fill mode on a variable. When this mode is on, fill values will not be written for the variable. This is helpful in high performance applications. For netCDF-4/HDF5 files (whether classic model or not), this may only be changed after the variable is defined, but before it is committed to disk (i.e. before the first NF_ENDDEF after the NF_DEF_VAR.) For classic and 64-bit offset file, the no_fill mode may be turned on and off at any time.

`FILL_VALUE`
A value which will be used as the fill value for the variable. Must be the same type as the variable. This will be written to a _FillValue attribute, created for this purpose. If NULL, this argument will be ignored.

Return Codes

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_ENOTVAR`
Can't find this variable.

`NF_ELATEDEF`
This variable has already been the subject of a NF_ENDDEF call. In netCDF-4 files NF_ENDDEF will be called automatically for any data read or write. Once enddef has been called, it is impossible to set the fill for a variable.

`NF_ENOTINDEFINE`
Not in define mode. This is returned for netCDF classic or 64-bit offset files, or for netCDF-4 files, when they were been created with NF_STRICT_NC3 flag. (see Section 2.5 [NF_CREATE], page 9).

`NF_EPERM` Attempt to create object in read-only file.

Example

6.9 Learn About Fill Parameters for a Variable: `NF_INQ_VAR_FILL`

The function NF_INQ_VAR_FILL returns the fill settings for a variable in a netCDF-4 file.

Usage

```
NF_INQ_VAR_FILL(INTEGER NCID, INTEGER VARID, INTEGER NO_FILL, FILL_VALUE)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`NO_FILL` An integer which will get a 1 if no_fill mode is set for this variable, and a zero if it is not set

`FILL_VALUE`
This will get the fill value for this variable. This parameter will be ignored if it is NULL.

Return Codes

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_ENOTVAR`
Can't find this variable.

Example

6.10 Define Compression Parameters for a Variable: `NF_DEF_VAR_DEFLATE`

The function NF_DEF_VAR_DEFLATE sets the deflate parameters for a variable in a netCDF-4 file.

When using parallel I/O for writing data, deflate cannot be used. This is because the compression makes it impossible for the HDF5 library to exactly map the data to disk location.

(Deflated data can be read with parallel I/O).

NF_DEF_VAR_DEFLATE must be called after the variable is defined, but before NF_ENDDEF is called.

Usage

```
NF_DEF_VAR_DEFLATE(INTEGER NCID, INTEGER VARID, INTEGER SHUFFLE, INTEGER DEFLATE,
                   INTEGER DEFLATE_LEVEL);
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`SHUFFLE` If non-zero, turn on the shuffle filter.

`DEFLATE` If non-zero, turn on the deflate filter at the level specified by the deflate_level parameter.

`DEFLATE_LEVEL`
Must be between 0 (no deflate, the default) and 9 (slowest, but "best" deflate).

If set to zero, no deflation takes place and the def_var_deflate call is ignored. This is slightly different from HDF5 handing of 0 deflate, which turns on the filter but makes only trivial changes to the data.

Informal testing at NetCDF World Headquarters suggests that there is little to be gained (with the limited set of test data used here), in setting the deflate level above 2 or 3.

Errors

NF_DEF_VAR_DEFLATE returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_ENOTVAR`
Can't find this variable.

`NF_ELATEDEF`
This variable has already been the subject of a NF_ENDDEF call. In netCDF-4 files NF_ENDDEF will be called automatically for any data read or write. Once enddef has been called, it is impossible to set the deflate for a variable.

`NF_ENOTINDEFINE`
Not in define mode. This is returned for netCDF classic or 64-bit offset files, or for netCDF-4 files, when they were been created with NF_STRICT_NC3 flag. (see Section 2.5 [NF_CREATE], page 9).

`NF_EPERM` Attempt to create object in read-only file.

`NF_EINVAL`
Invalid deflate_level. The deflate level must be between 0 and 9, inclusive.

Example

In this example from nf_test/ftst_vars.F, a file is created with two dimensions and one variable. Chunking, deflate, and the fletcher32 filter are turned on. The deflate level is set to 4 below.

```
C     Create the netCDF file.
      retval = nf_create(FILE_NAME, NF_NETCDF4, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the dimensions.
      retval = nf_def_dim(ncid, "x", NX, x_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, "y", NY, y_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the variable.
      dimids(1) = y_dimid
      dimids(2) = x_dimid
      retval = NF_DEF_VAR(ncid, "data", NF_INT, NDIMS, dimids, varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Turn on chunking.
      chunks(1) = NY
      chunks(2) = NX
      retval = NF_DEF_VAR_CHUNKING(ncid, varid, NF_CHUNKED, chunks)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Turn on deflate compression, fletcher32 checksum.
      retval = NF_DEF_VAR_deflate(ncid, varid, 0, 1, 4)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = NF_DEF_VAR_FLETCHER32(ncid, varid, NF_FLETCHER32)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

6.11 Learn About Deflate Parameters for a Variable: NF_INQ_VAR_DEFLATE

The function NF_INQ_VAR_DEFLATE returns the deflate settings for a variable in a netCDF-4 file.

It is not necessary to know the deflate settings to read the variable. (Deflate is completely transparent to readers of the data).

Usage

```
NF_INQ_VAR_DEFLATE(INTEGER NCID, INTEGER VARID, INTEGER SHUFFLE,
                   INTEGER DEFLATE, INTEGER DEFLATE_LEVEL);
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`SHUFFLE` NF_INQ_VAR_DEFLATE will set this to a 1 if the shuffle filter is turned on for this variable, and a 0 otherwise.

`DEFLATE` NF_INQ_VAR_DEFLATE will set this to a 1 if the deflate filter is turned on for this variable, and a 0 otherwise.

`DEFLATE_LEVEL`
NF_INQ_VAR_DEFLATE function will write the deflate_level here, if deflate is in use.

Errors

NF_INQ_VAR_DEFLATE returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_ENOTVAR`
Can't find this variable.

Example

In this example code from nf_test/ftst_vars.F, a file with a variable using deflate is opened, and the deflate level checked.

```
C     Is everything set that is supposed to be?
      retval = nf_inq_var_deflate(ncid, varid, shuffle, deflate,
     +     deflate_level)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (shuffle .ne. 0 .or. deflate .ne. 1 .or.
     +     deflate_level .ne. 4) stop 2
```

6.12 Learn About Szip Parameters for a Variable: `NF_INQ_VAR_SZIP`

The function NF_INQ_VAR_SZIP returns the szip settings for a variable in a netCDF-4 file.

It is not necessary to know the szip settings to read the variable. (Szip is completely transparent to readers of the data).

Usage

```
NF_INQ_VAR_SZIP(INTEGER NCID, INTEGER VARID, INTEGER OPTION_MASK,
                              PIXELS_PER_BLOCK);
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`OPTION_MASK`
This will be set to the option_mask value.

`PIXELS_PER_BLOCK`
The number of bits per pixel will be put here.

Errors

NF_INQ_VAR_SZIP returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_ENOTVAR`
Can't find this variable.

Example

6.13 Define Checksum Parameters for a Variable: `NF_DEF_VAR_FLETCHER32`

The function NF_DEF_VAR_FLETCHER32 sets the checksum property for a variable in a netCDF-4 file.

This function may only be called after the variable is defined, but before NF_ENDDEF is called.

Usage

```
NF_DEF_VAR_FLETCHER32(INTEGER NCID, INTEGER VARID, INTEGER CHECKSUM);
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`CHECKSUM` If this is NF_FLETCHER32, fletcher32 checksums will be turned on for this variable.

Errors

NF_DEF_VAR_FLETCHER32 returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
: Not a netCDF-4 file.

`NF_ENOTVAR`
: Can't find this variable.

`NF_ELATEDEF`
: This variable has already been the subject of a NF_ENDDEF call. In netCDF-4 files NF_ENDDEF will be called automatically for any data read or write. Once enddef has been called, it is impossible to set the checksum property for a variable.

`NF_ENOTINDEFINE`
: Not in define mode. This is returned for netCDF classic or 64-bit offset files, or for netCDF-4 files, when they were been created with NF_STRICT_NC3 flag. (see Section 2.5 [NF_CREATE], page 9).

`NF_EPERM` Attempt to create object in read-only file.

Example

In this example from nf_test/ftst_vars.F, the variable in a file has the Fletcher32 checksum filter turned on.

```
C     Create the netCDF file.
      retval = nf_create(FILE_NAME, NF_NETCDF4, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the dimensions.
      retval = nf_def_dim(ncid, "x", NX, x_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, "y", NY, y_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the variable.
      dimids(1) = y_dimid
      dimids(2) = x_dimid
      retval = NF_DEF_VAR(ncid, "data", NF_INT, NDIMS, dimids, varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Turn on chunking.
```

```
      chunks(1) = NY
      chunks(2) = NX
      retval = NF_DEF_VAR_CHUNKING(ncid, varid, NF_CHUNKED, chunks)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Turn on deflate compression, fletcher32 checksums.
      retval = NF_DEF_VAR_DEFLATE(ncid, varid, 0, 1, 4)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = NF_DEF_VAR_FLETCHER32(ncid, varid, NF_FLETCHER32)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

6.14 Learn About Checksum Parameters for a Variable: NF_INQ_VAR_FLETCHER32

The function NF_INQ_VAR_FLETCHER32 returns the checksum settings for a variable in a netCDF-4 file.

Usage

```
NF_INQ_VAR_FLETCHER32(INTEGER NCID, INTEGER VARID, INTEGER CHECKSUM);
```

NCID NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

VARID Variable ID.

CHECKSUM NF_INQ_VAR_FLETCHER32 will set this to NF_FLETCHER32 if the fletcher32 filter is turned on for this variable, and NF_NOCHECKSUM if it is not.

Errors

NF_INQ_VAR_FLETCHER32 returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

NF_NOERR No error.

NF_BADID Bad ncid.

NF_ENOTNC4
Not a netCDF-4 file.

NF_ENOTVAR
Can't find this variable.

Example

In this example from nf_test/ftst_vars.F the checksum filter is checked for a file. Since it was turned on for this variable, the checksum variable is set to NF_FLETCHER32.

```
      retval = nf_inq_var_fletcher32(ncid, varid, checksum)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (checksum .ne. NF_FLETCHER32) stop 2
```

6.15 Define Endianness of a Variable: `NF_DEF_VAR_ENDIAN`

The function NF_DEF_VAR_ENDIAN sets the endianness for a variable in a netCDF-4 file.

This function must be called after the variable is defined, but before NF_ENDDEF is called.

By default, netCDF-4 variables are in native endianness. That is, they are big-endian on a big-endian machine, and little-endian on a little endian machine.

In some cases a user might wish to change from native endianness to either big or little-endianness. This function allows them to do that.

Usage

```
NF_DEF_VAR_ENDIAN(INTEGER NCID, INTEGER VARID, INTEGER ENDIAN)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`ENDIAN` Set to NF_ENDIAN_NATIVE for native endianness. (This is the default). Set to NF_ENDIAN_LITTLE for little endian, or NF_ENDIAN_BIG for big endian.

Errors

NF_DEF_VAR_ENDIAN returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_ENOTVAR`
Can't find this variable.

`NF_ELATEDEF`
This variable has already been the subject of a NF_ENDDEF call. In netCDF-4 files NF_ENDDEF will be called automatically for any data read or write. Once enddef has been called, it is impossible to set the endianness of a variable.

`NF_ENOTINDEFINE`
Not in define mode. This is returned for netCDF classic or 64-bit offset files, or for netCDF-4 files, when they were been created with NF_STRICT_NC3 flag, and the file is not in define mode. (see Section 2.5 [NF_CREATE], page 9).

`NF_EPERM` Attempt to create object in read-only file.

Example

In this example from nf_test/ftst_vars.c, a file is created with one variable, and its endianness is set to NF_ENDIAN_BIG.

```
C     Create the netCDF file.
      retval = nf_create(FILE_NAME, NF_NETCDF4, ncid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the dimensions.
      retval = nf_def_dim(ncid, "x", NX, x_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)
      retval = nf_def_dim(ncid, "y", NY, y_dimid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Define the variable.
      dimids(1) = y_dimid
      dimids(2) = x_dimid
      retval = NF_DEF_VAR(ncid, "data", NF_INT, NDIMS, dimids, varid)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Turn on chunking.
      chunks(1) = NY
      chunks(2) = NX
      retval = NF_DEF_VAR_chunking(ncid, varid, 0, chunks)
      if (retval .ne. nf_noerr) call handle_err(retval)

C     Set variable to big-endian (default is whatever is native to
C     writing machine).
      retval = NF_DEF_VAR_endian(ncid, varid, NF_ENDIAN_BIG)
      if (retval .ne. nf_noerr) call handle_err(retval)
```

6.16 Learn About Endian Parameters for a Variable: `NF_INQ_VAR_ENDIAN`

The function NF_INQ_VAR_ENDIAN returns the endianness settings for a variable in a netCDF-4 file.

Usage

```
NF_INQ_VAR_ENDIAN(INTEGER NCID, INTEGER VARID, INTEGER ENDIAN)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`ENDIAN` NF_INQ_VAR_ENDIAN will set this to NF_ENDIAN_LITTLE if this variable is stored in little-endian format, NF_ENDIAN_BIG if it is stored in big-endian format, and NF_ENDIAN_NATIVE if the endianness is not set, and the variable is not created yet.

Errors

NF_INQ_VAR_ENDIAN returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error.

Possible return codes include:

`NF_NOERR` No error.

`NF_BADID` Bad ncid.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_ENOTVAR`
Can't find this variable.

Example

In this example from nf_test/ftst_vars.F, the endianness of a variable is checked to make sure it is NF_ENDIAN_BIG.

```
      retval = nf_inq_var_endian(ncid, varid, endianness)
      if (retval .ne. nf_noerr) call handle_err(retval)
      if (endianness .ne. NF_ENDIAN_BIG) stop 2
```

6.17 Get a Variable ID from Its Name: NF_INQ_VARID

The function NF_INQ_VARID returns the ID of a netCDF variable, given its name.

Usage

```
INTEGER FUNCTION NF_INQ_VARID(INTEGER NCID, CHARACTER*(*) NAME,
                              INTEGER varid)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`NAME` Variable name for which ID is desired.

`varid` Returned variable ID.

Errors

NF_INQ_VARID returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified variable name is not a valid name for a variable in the specified netCDF dataset.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_INQ_VARID to find out the ID of a variable named rh in an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  STATUS, NCID, RHID
   ...
STATUS = NF_OPEN ('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.18 Get Information about a Variable from Its ID: NF_INQ_VAR family

A family of functions that returns information about a netCDF variable, given its ID. Information about a variable includes its name, type, number of dimensions, a list of dimension IDs describing the shape of the variable, and the number of variable attributes that have been assigned to the variable.

The function NF_INQ_VAR returns all the information about a netCDF variable, given its ID. The other functions each return just one item of information about a variable.

These other functions include NF_INQ_VARNAME, NF_INQ_VARTYPE, NF_INQ_VARNDIMS, NF_INQ_VARDIMID, and NF_INQ_VARNATTS.

Usage

```
INTEGER FUNCTION NF_INQ_VAR      (INTEGER NCID, INTEGER VARID,
                                  CHARACTER*(*) name, INTEGER xtype,
                                  INTEGER ndims, INTEGER dimids(*),
                                  INTEGER natts)
INTEGER FUNCTION NF_INQ_VARNAME  (INTEGER NCID, INTEGER VARID,
                                  CHARACTER*(*) name)
INTEGER FUNCTION NF_INQ_VARTYPE  (INTEGER NCID, INTEGER VARID,
                                  INTEGER xtype)
INTEGER FUNCTION NF_INQ_VARNDIMS (INTEGER NCID, INTEGER VARID,
                                  INTEGER ndims)
INTEGER FUNCTION NF_INQ_VARDIMID (INTEGER NCID, INTEGER VARID,
                                  INTEGER dimids(*))
INTEGER FUNCTION NF_INQ_VARNATTS (INTEGER NCID, INTEGER VARID,
                                  INTEGER natts)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`NAME` Returned variable name. The caller must allocate space for the returned name. The maximum possible length, in characters, of a variable name is given by the predefined constant NF_MAX_NAME.

`xtype` Returned variable type, one of the set of predefined netCDF external data types. The type of this parameter, NF_TYPE, is defined in the netCDF header file.

The valid netCDF external data types are NF_BYTE, NF_CHAR, NF_SHORT, NF_INT, NF_FLOAT, AND NF_DOUBLE.

`ndims` Returned number of dimensions the variable was defined as using. For example, 2 indicates a matrix, 1 indicates a vector, and 0 means the variable is a scalar with no dimensions.

`dimids` Returned vector of *ndimsp dimension IDs corresponding to the variable dimensions. The caller must allocate enough space for a vector of at least *ndimsp integers to be returned. The maximum possible number of dimensions for a variable is given by the predefined constant NF_MAX_VAR_DIMS.

`natts` Returned number of variable attributes assigned to this variable.

These functions return the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_INQ_VAR to find out about a variable named rh in an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  STATUS, NCID
INTEGER  RHID               ! variable ID
CHARACTER*31 RHNAME         ! variable name
INTEGER  RHTYPE             ! variable type
INTEGER  RHN                ! number of dimensions
INTEGER  RHDIMS(NF_MAX_VAR_DIMS)   ! variable shape
INTEGER  RHNATT                    ! number of attributes
   ...
STATUS = NF_OPEN ('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)  ! get ID
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_INQ_VAR (NCID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.19 Write a Single Data Value: NF_PUT_VAR1_ type

The functions NF_PUT_VAR1_type (for various types) put a single data value of the specified type into a variable of an open netCDF dataset that is in data mode. Inputs are the netCDF ID, the variable ID, an index that specifies which value to add or alter, and the data value. The value is converted to the external data type of the variable, if necessary.

Usage

```
INTEGER FUNCTION  NF_PUT_VAR1_TEXT(INTEGER NCID, INTEGER VARID,
                                   INTEGER INDEX(*), CHARACTER CHVAL)
INTEGER FUNCTION  NF_PUT_VAR1_INT1(INTEGER NCID, INTEGER VARID,
                                   INTEGER INDEX(*), INTEGER*1 I1VAL)
INTEGER FUNCTION  NF_PUT_VAR1_INT2(INTEGER NCID, INTEGER VARID,
                                   INTEGER INDEX(*), INTEGER*2 I2VAL)
INTEGER FUNCTION  NF_PUT_VAR1_INT (INTEGER NCID, INTEGER VARID,
                                   INTEGER INDEX(*), INTEGER   IVAL)
INTEGER FUNCTION  NF_PUT_VAR1_REAL(INTEGER NCID, INTEGER VARID,
                                   INTEGER INDEX(*), REAL      RVAL)
INTEGER FUNCTION  NF_PUT_VAR1_DOUBLE(INTEGER NCID, INTEGER VARID,
                                   INTEGER INDEX(*), DOUBLE    DVAL)
INTEGER FUNCTION  NF_PUT_VAR1(INTEGER NCID, INTEGER VARID,
                                   INTEGER INDEX(*), *)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`INDEX` The index of the data value to be written. The indices are relative to 1, so for example, the first data value of a two-dimensional variable would have index (1,1). The elements of index must correspond to the variable's dimensions. Hence, if the variable uses the unlimited dimension, the last index would correspond to the record number.

`CHVAL`
`I1VAL`
`I2VAL`
`IVAL`
`RVAL`
`DVAL` Pointer to the data value to be written. If the type of data values differs from the netCDF variable type, type conversion will occur. See Section "Type Conversion" in *The NetCDF Users Guide*.

Errors

NF_PUT_VAR1_ type returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified indices were out of range for the rank of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension length will cause an error.
- The specified value is out of the range of values representable by the external data type of the variable.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_PUT_VAR1_DOUBLE to set the (4,3,2) element of the variable named rh to 0.5 in an existing netCDF dataset named foo.nc. For simplicity in this example, we assume that we know that rh is dimensioned with lon, lat, and time, so we want to set the value of rh that corresponds to the fourth lon value, the third lat value, and the second time value:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  STATUS            ! error status
INTEGER  NCID
INTEGER  RHID              ! variable ID
INTEGER  RHINDX(3)         ! where to put value
DATA RHINDX /4, 3, 2/
   ...
STATUS = NF_OPEN ('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)  ! get ID
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_PUT_VAR1_DOUBLE (NCID, RHID, RHINDX, 0.5)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.20 Write an Entire Variable: NF_PUT_VAR_ *type*

The NF_PUT_VAR_ *type* family of functions write all the values of a variable into a netCDF variable of an open netCDF dataset. This is the simplest interface to use for writing a value in a scalar variable or whenever all the values of a multidimensional variable can all be written at once. The values to be written are associated with the netCDF variable by assuming that the last dimension of the netCDF variable varies fastest in the C interface. The values are converted to the external data type of the variable, if necessary.

Take care when using the simplest forms of this interface with record variables (variables that use the NF_UNLIMITED dimension) when you don't specify how many records are to be written. If you try to write all the values of a record variable into a netCDF file that has no record data yet (hence has 0 records), nothing will be written. Similarly, if you try to write all the values of a record variable from an array but there are more records in the file than you assume, more in-memory data will be accessed than you expect, which may cause a segmentation violation. To avoid such problems, it is better to use the NF_PUT_VARA_type interfaces for variables that use the NF_UNLIMITED dimension. See Section 6.21 [NF_PUT_VARA_ type], page 93.

Usage

```
INTEGER FUNCTION NF_PUT_VAR_TEXT  (INTEGER NCID, INTEGER VARID,
                                   CHARACTER*(*) TEXT)
INTEGER FUNCTION NF_PUT_VAR_INT1  (INTEGER NCID, INTEGER VARID,
                                   INTEGER*1 I1VALS(*))
INTEGER FUNCTION NF_PUT_VAR_INT2  (INTEGER NCID, INTEGER VARID,
```

```
                                   INTEGER*2 I2VALS(*))
INTEGER FUNCTION NF_PUT_VAR_INT   (INTEGER NCID, INTEGER VARID,
                                   INTEGER IVALS(*))
INTEGER FUNCTION NF_PUT_VAR_REAL  (INTEGER NCID, INTEGER VARID,
                                   REAL RVALS(*))
INTEGER FUNCTION NF_PUT_VAR_DOUBLE(INTEGER NCID, INTEGER VARID,
                                   DOUBLE DVALS(*))
INTEGER FUNCTION NF_PUT_VAR       (INTEGER NCID, INTEGER VARID,
                                   VALS(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`TEXT`
`I1VALS`
`I2VALS`
`IVALS`
`RVALS`
`DVALS`
`VALS` The block of data values to be written. The data should be of the type appropriate for the function called. You cannot put CHARACTER data into a numeric variable or numeric data into a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur (see Section "Type Conversion" in *The NetCDF Users Guide*). The order in which the data will be written into the specified variable is with the first dimension varying fastest (like the ordinary FORTRAN convention).

Errors

Members of the NF_PUT_VAR_ *type* family return the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- One or more of the specified values are out of the range of values representable by the external data type of the variable.
- One or more of the specified values are out of the range of values representable by the external data type of the variable.
- The specified netCDF dataset is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_PUT_VAR_DOUBLE to add or change all the values of the variable named rh to 0.5 in an existing netCDF dataset named foo.nc. For simplicity in this example, we assume that we know that rh is dimensioned with lon and lat, and that there are ten lon values and five lat values.

```
INCLUDE 'netcdf.inc'
```

```
   ...
PARAMETER (LATS=5, LONS=10) ! dimension lengths
INTEGER  STATUS, NCID
INTEGER  RHID                        ! variable ID
DOUBLE RHVALS(LONS, LATS)
   ...
STATUS = NF_OPEN ('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
DO 10 ILON = 1, LONS
   DO 10 ILAT = 1, LATS
         RHVALS(ILON, ILAT) = 0.5
10 CONTINUE
STATUS = NF_PUT_var_DOUBLE (NCID, RHID, RHVALS)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.21 Write an Array of Values: NF_PUT_VARA_ *type*

The function NF_PUT_VARA_ *type* writes values into a netCDF variable of an open netCDF dataset. The part of the netCDF variable to write is specified by giving a corner and a vector of edge lengths that refer to an array section of the netCDF variable. The values to be written are associated with the netCDF variable by assuming that the first dimension of the netCDF variable varies fastest in the FORTRAN interface. The netCDF dataset must be in data mode.

Usage

```
INTEGER FUNCTION NF_PUT_VARA_TEXT(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  CHARACTER*(*) TEXT)
INTEGER FUNCTION NF_PUT_VARA_INT1(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  INTEGER*1 I1VALS(*))
INTEGER FUNCTION NF_PUT_VARA_INT2(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  INTEGER*2 I2VALS(*))
INTEGER FUNCTION NF_PUT_VARA_INT (INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  INTEGER IVALS(*))
INTEGER FUNCTION NF_PUT_VARA_REAL(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  REAL RVALS(*))
INTEGER FUNCTION NF_PUT_VARA_DOUBLE(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  DOUBLE DVALS(*))
INTEGER FUNCTION NF_PUT_VARA     (INTEGER NCID, INTEGER VARID,
```

```
                                INTEGER START(*), INTEGER COUNT(*),
                                VALS(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`START` A vector of integers specifying the index in the variable where the first of the data values will be written. The indices are relative to 1, so for example, the first data value of a variable would have index (1, 1, ..., 1). The length of START must be the same as the number of dimensions of the specified variable. The elements of START must correspond to the variable's dimensions in order. Hence, if the variable is a record variable, the last index would correspond to the starting record number for writing the data values.

`COUNT` A vector of integers specifying the edge lengths along each dimension of the block of data values to written. To write a single value, for example, specify COUNT as (1, 1, ..., 1). The length of COUNT is the number of dimensions of the specified variable. The elements of COUNT correspond to the variable's dimensions. Hence, if the variable is a record variable, the last element of COUNT corresponds to a count of the number of records to write.

Note: setting any element of the count array to zero causes the function to exit without error, and without doing anything.

`TEXT`
`I1VALS`
`I2VALS`
`IVALS`
`RVALS`
`DVALS`
`VALS` The block of data values to be written. The data should be of the type appropriate for the function called. You cannot put CHARACTER data into a numeric variable or numeric data into a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur (see Section "Type Conversion" in *The NetCDF Users Guide*).

Errors

NF_PUT_VARA_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified corner indices were out of range for the rank of the specified variable. For example, a negative index, or an index that is larger than the corresponding dimension length will cause an error.
- The specified edge lengths added to the specified corner would have referenced data out of range for the rank of the specified variable. For example, an edge length that is larger than the corresponding dimension length minus the corner index will cause an error.
- One or more of the specified values are out of the range of values representable by the external data type of the variable.

- The specified netCDF dataset is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_PUT_VARA_DOUBLE to add or change all the values of the variable named rh to 0.5 in an existing netCDF dataset named foo.nc. For simplicity in this example, we assume that we know that rh is dimensioned with time, lat, and lon, and that there are three time values, five lat values, and ten lon values.

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIMS=3)         ! number of dimensions
PARAMETER (TIMES=3, LATS=5, LONS=10) ! dimension lengths
INTEGER  STATUS, NCID, TIMES
INTEGER  RHID               ! variable ID
INTEGER  START(NDIMS), COUNT(NDIMS)
DOUBLE RHVALS(LONS, LATS, TIMES)
DATA START /1, 1, 1/        ! start at first value
DATA COUNT /LONS, LATS, TIMES/
   ...
STATUS = NF_OPEN ('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
DO 10 ILON = 1, LONS
   DO 10 ILAT = 1, LATS
      DO 10 ITIME = 1, TIMES
         RHVALS(ILON, ILAT, ITIME) = 0.5
10 CONTINUE
STATUS = NF_PUT_VARA_DOUBLE (NCID, RHID, START, COUNT, RHVALS)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.22 NF_PUT_VARS_ *type*

Each member of the family of functions NF_PUT_VARS_ *type* writes a subsampled (strided) array section of values into a netCDF variable of an open netCDF dataset. The subsampled array section is specified by giving a corner, a vector of counts, and a stride vector. The netCDF dataset must be in data mode.

Usage

```
INTEGER FUNCTION NF_PUT_VARS_TEXT  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*),CHARACTER*(*) TEXT)
INTEGER FUNCTION NF_PUT_VARS_INT1  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*),INTEGER*1 I1VALS(*))
```

```
INTEGER FUNCTION NF_PUT_VARS_INT2  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*),INTEGER*2 I2VALS(*))
INTEGER FUNCTION NF_PUT_VARS_INT   (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IVALS(*))
INTEGER FUNCTION NF_PUT_VARS_REAL  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*),  REAL RVALS(*))
INTEGER FUNCTION NF_PUT_VARS_DOUBLE(INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*),  DOUBLE DVALS(*))
INTEGER FUNCTION NF_PUT_VARS       (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), VALS(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`START` A vector of integers specifying the index in the variable where the first of the data values will be written. The indices are relative to 1, so for example, the first data value of a variable would have index (1, 1, ..., 1). The elements of START correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the starting record number for writing the data values.

`COUNT` A vector of integers specifying the number of indices selected along each dimension. To write a single value, for example, specify COUNT as (1, 1, ..., 1). The elements of COUNT correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last element of COUNT corresponds to a count of the number of records to write.

Note: setting any element of the count array to zero causes the function to exit without error, and without doing anything.

`STRIDE` A vector of integers that specifies the sampling interval along each dimension of the netCDF variable. The elements of the stride vector correspond, in order, to the netCDF variable's dimensions (STRIDE(1) gives the sampling interval along the most rapidly varying dimension of the netCDF variable). Sampling intervals are specified in type-independent units of elements (a value of 1 selects consecutive elements of the netCDF variable along the corresponding dimension, a value of 2 selects every other element, etc.).

`TEXT`
`I1VALS`
`I2VALS`
`IVALS`
`RVALS`
`DVALS`
`VALS` The block of data values to be written. The data should be of the type appropriate for the function called. You cannot put CHARACTER data into a

numeric variable or numeric data into a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur (see Section "Type Conversion" in *The NetCDF Users Guide*).

Errors

NF_PUT_VARS_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified start, count and stride generate an index which is out of range.
- One or more of the specified values are out of the range of values representable by the external data type of the variable.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example of using NF_PUT_VARS_REAL to write – from an internal array – every other point of a netCDF variable named rh which is described by the FORTRAN declaration REAL RH(6,4) (note the size of the dimensions):

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIM=2)   ! rank of netCDF variable
INTEGER NCID         ! netCDF dataset ID
INTEGER STATUS       ! return code
INTEGER RHID         ! variable ID
INTEGER START(NDIM)  ! netCDF variable start point
INTEGER COUNT(NDIM)  ! size of internal array
INTEGER STRIDE(NDIM) ! netCDF variable subsampling intervals
REAL RH(3,2)         ! note subsampled sizes for netCDF variable
                     ! dimensions
DATA START   /1, 1/  ! start at first netCDF variable value
DATA COUNT   /3, 2/  ! size of internal array: entire (subsampled)
                     ! netCDF variable
DATA STRIDE /2, 2/   ! access every other netCDF element
   ...
STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID(NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_PUT_VARS_REAL(NCID, RHID, START, COUNT, STRIDE, RH)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.23 NF_PUT_VARM_ *type*

The NF_PUT_VARM_ *type* family of functions writes a mapped array section of values into a netCDF variable of an open netCDF dataset. The mapped array section is specified by giving a corner, a vector of counts, a stride vector, and an index mapping vector. The index mapping vector is a vector of integers that specifies the mapping between the dimensions of a netCDF variable and the in-memory structure of the internal data array. No assumptions are made about the ordering or length of the dimensions of the data array. The netCDF dataset must be in data mode.

Usage

```
INTEGER FUNCTION NF_PUT_VARM_TEXT  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            CHARACTER*(*) TEXT)
INTEGER FUNCTION NF_PUT_VARM_INT1  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            INTEGER*1 I1VALS(*))
INTEGER FUNCTION NF_PUT_VARM_INT2  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            INTEGER*2 I2VALS(*))
INTEGER FUNCTION NF_PUT_VARM_INT   (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            INTEGER IVALS(*))
INTEGER FUNCTION NF_PUT_VARM_REAL  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            REAL RVALS(*))
INTEGER FUNCTION NF_PUT_VARM_DOUBLE(INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            DOUBLE DVALS(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`START` A vector of integers specifying the index in the variable where the first of the data values will be written. The indices are relative to 1, so for example, the first data value of a variable would have index (1, 1, ..., 1). The elements of START correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the starting record number for writing the data values.

`COUNT` A vector of integers specifying the number of indices selected along each dimension. To write a single value, for example, specify COUNT as (1, 1, ..., 1). The elements of COUNT correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last element of COUNT corresponds to a count of the number of records to write.

Note: setting any element of the count array to zero causes the function to exit without error, and without doing anything.

`STRIDE` A vector of integers that specifies the sampling interval along each dimension of the netCDF variable. The elements of the stride vector correspond, in order, to the netCDF variable's dimensions (STRIDE(1) gives the sampling interval along the most rapidly varying dimension of the netCDF variable). Sampling intervals are specified in type-independent units of elements (a value of 1 selects consecutive elements of the netCDF variable along the corresponding dimension, a value of 2 selects every other element, etc.).

`IMAP` A vector of integers that specifies the mapping between the dimensions of a netCDF variable and the in-memory structure of the internal data array. The elements of the index mapping vector correspond, in order, to the netCDF variable's dimensions (IMAP(1) gives the distance between elements of the internal array corresponding to the most rapidly varying dimension of the netCDF variable). Distances between elements are specified in units of elements (the distance between internal elements that occupy adjacent memory locations is 1 and not the element's byte-length as in netCDF 2).

`TEXT`
`I1VALS`
`I2VALS`
`IVALS`
`RVALS`
`DVALS` The data values to be written. The data should be of the type appropriate for the function called. You cannot put CHARACTER data into a numeric variable or numeric data into a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur (see Section "Type Conversion" in *The NetCDF Users Guide*).

Errors

NF_PUT_VARM_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified START, COUNT, and STRIDE generate an index which is out of range. Note that no error checking is possible on the imap vector.
- One or more of the specified values are out of the range of values representable by the external data type of the variable.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

The following IMAP vector maps in the trivial way a 2x3x4 netCDF variable and an internal array of the same shape:

```
REAL A(2,3,4)       ! same shape as netCDF variable
INTEGER IMAP(3)
DATA IMAP /1, 2, 6/ ! netCDF dimension       inter-element distance
                    ! ----------------       ----------------------
                    ! most rapidly varying       1
                    ! intermediate               2 (=IMAP(1)*2)
                    ! most slowly varying        6 (=IMAP(2)*3)
```

Using the IMAP vector above with NF_PUT_VARM_REAL obtains the same result as simply using NF_PUT_VAR_REAL.

Here is an example of using NF_PUT_VARM_REAL to write – from a transposed, internal array – a netCDF variable named rh which is described by the FORTRAN declaration REAL RH(4,6) (note the size and order of the dimensions):

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIM=2)   ! rank of netCDF variable
INTEGER NCID         ! netCDF ID
INTEGER STATUS       ! return code
INTEGER RHID         ! variable ID
INTEGER START(NDIM)  ! netCDF variable start point
INTEGER COUNT(NDIM)  ! size of internal array
INTEGER STRIDE(NDIM) ! netCDF variable subsampling intervals
INTEGER IMAP(NDIM)   ! internal array inter-element distances
REAL RH(6,4)         ! note transposition of netCDF variable dimensions
DATA START   /1, 1/  ! start at first netCDF variable element
DATA COUNT   /4, 6/  ! entire netCDF variable; order corresponds
                     ! to netCDF variable -- not internal array
DATA STRIDE /1, 1/   ! sample every netCDF element
DATA IMAP   /6, 1/   ! would be /1, 4/ if not transposing

STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID(NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_PUT_VARM_REAL(NCID, RHID, START, COUNT, STRIDE, IMAP, RH)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

Here is another example of using NF_PUT_VARM_REAL to write – from a transposed, internal array – a subsample of the same netCDF variable, by writing every other point of the netCDF variable:

```
INCLUDE 'netcdf.inc'
   ...
```

```
PARAMETER (NDIM=2)   ! rank of netCDF variable
INTEGER NCID         ! netCDF dataset ID
INTEGER STATUS       ! return code
INTEGER RHID         ! variable ID
INTEGER START(NDIM)  ! netCDF variable start point
INTEGER COUNT(NDIM)  ! size of internal array
INTEGER STRIDE(NDIM) ! netCDF variable subsampling intervals
INTEGER IMAP(NDIM)   ! internal array inter-element distances
REAL RH(3,2)         ! note transposition of (subsampled) dimensions
DATA START   /1, 1/  ! start at first netCDF variable value
DATA COUNT   /2, 3/  ! order of (subsampled) dimensions corresponds
                     ! to netCDF variable -- not internal array
DATA STRIDE /2, 2/   ! sample every other netCDF element
DATA IMAP   /3, 1/   ! would be ‘1, 2’ if not transposing
   ...
STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID(NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_PUT_VARM_REAL(NCID, RHID, START, COUNT, STRIDE, IMAP, RH)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.24 NF_GET_VAR1_ *type*

The functions NF_GET_VAR1_ *type* get a single data value from a variable of an open netCDF dataset that is in data mode. Inputs are the netCDF ID, the variable ID, a multidimensional index that specifies which value to get, and the address of a location into which the data value will be read. The value is converted from the external data type of the variable, if necessary.

Usage

```
INTEGER FUNCTION  NF_GET_VAR1_TEXT(INTEGER NCID, INTEGER VARID,
                                 INTEGER INDEX(*), CHARACTER CHVAL)
INTEGER FUNCTION  NF_GET_VAR1_INT1(INTEGER NCID, INTEGER VARID,
                                 INTEGER INDEX(*), INTEGER*1 I1VAL)
INTEGER FUNCTION  NF_GET_VAR1_INT2(INTEGER NCID, INTEGER VARID,
                                 INTEGER INDEX(*), INTEGER*2 I2VAL)
INTEGER FUNCTION  NF_GET_VAR1_INT (INTEGER NCID, INTEGER VARID,
                                 INTEGER INDEX(*), INTEGER   IVAL)
INTEGER FUNCTION  NF_GET_VAR1_REAL(INTEGER NCID, INTEGER VARID,
                                 INTEGER INDEX(*), REAL      RVAL)
INTEGER FUNCTION  NF_GET_VAR1_DOUBLE(INTEGER NCID, INTEGER VARID,
                                 INTEGER INDEX(*), DOUBLE    DVAL)
INTEGER FUNCTION  NF_GET_VAR1(INTEGER NCID, INTEGER VARID,
                                 INTEGER INDEX(*), VAL)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`INDEX` The index of the data value to be read. The indices are relative to 1, so for example, the first data value of a two-dimensional variable has index (1,1). The elements of index correspond to the variable's dimensions. Hence, if the variable is a record variable, the last index is the record number.

`CHVAL`
`I1VAL`
`I2VAL`
`IVAL`
`RVAL`
`DVAL`
`VAL` The location into which the data value will be read. You cannot get CHARACTER data from a numeric variable or numeric data from a character variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur. (see Section "Type Conversion" in *The NetCDF Users Guide*).

Errors

NF_GET_VAR1_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified indices were out of range for the rank of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension length will cause an error.
- The value is out of the range of values representable by the desired data type.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_GET_VAR1_DOUBLE to get the (4,3,2) element of the variable named rh in an existing netCDF dataset named foo.nc. For simplicity in this example, we assume that we know that rh is dimensioned with lon, lat, and time, so we want to get the value of rh that corresponds to the fourth lon value, the third lat value, and the second time value:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID
INTEGER RHID           ! variable ID
INTEGER RHINDX(3)      ! where to get value
DOUBLE PRECISION RHVAL ! put it here
DATA RHINDX /4, 3, 2/
   ...
```

```
STATUS = NF_OPEN ('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_GET_VAR1_DOUBLE (NCID, RHID, RHINDX, RHVAL)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.25 NF_GET_VAR_ *type*

The members of the NF_GET_VAR_ *type* family of functions read all the values from a netCDF variable of an open netCDF dataset. This is the simplest interface to use for reading the value of a scalar variable or when all the values of a multidimensional variable can be read at once. The values are read into consecutive locations with the first dimension varying fastest. The netCDF dataset must be in data mode.

Take care when using the simplest forms of this interface with record variables (variables that use the NF_UNLIMITED dimension) when you don't specify how many records are to be read. If you try to read all the values of a record variable into an array but there are more records in the file than you assume, more data will be read than you expect, which may cause a segmentation violation. To avoid such problems, it is better to use the NF_GET_VARA_type interfaces for variables that use the NF_UNLIMITED dimension. See Section 6.26 [NF_GET_VARA_ type], page 104.

Usage

```
INTEGER FUNCTION NF_GET_VAR_TEXT  (INTEGER NCID, INTEGER VARID,
                                   CHARACTER*(*) text)
INTEGER FUNCTION NF_GET_VAR_INT1  (INTEGER NCID, INTEGER VARID,
                                   INTEGER*1 i1vals(*))
INTEGER FUNCTION NF_GET_VAR_INT2  (INTEGER NCID, INTEGER VARID,
                                   INTEGER*2 i2vals(*))
INTEGER FUNCTION NF_GET_VAR_INT   (INTEGER NCID, INTEGER VARID,
                                   INTEGER ivals(*))
INTEGER FUNCTION NF_GET_VAR_REAL  (INTEGER NCID, INTEGER VARID,
                                   REAL rvals(*))
INTEGER FUNCTION NF_GET_VAR_DOUBLE(INTEGER NCID, INTEGER VARID,
                                   DOUBLE dvals(*))
INTEGER FUNCTION NF_GET_VAR       (INTEGER NCID, INTEGER VARID,
                                   vals(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`TEXT`
`I1VALS`
`I2VALS`
`IVALS`
`RVALS`
`DVALS`
`VALS` The block of data values to be read. The data should be of the type appropriate for the function called. You cannot read CHARACTER data from a numeric variable or numeric data from a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur (see Section “Type Conversion” in *The NetCDF Users Guide*).

Errors

NF_GET_VAR_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- One or more of the values are out of the range of values representable by the desired type.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_GET_VAR_DOUBLE to read all the values of the variable named rh from an existing netCDF dataset named foo.nc. For simplicity in this example, we assume that we know that rh is dimensioned with lon and lat, and that there are ten lon values and five lat values.

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (LATS=5, LONS=10) ! dimension lengths
INTEGER STATUS, NCID
INTEGER RHID                         ! variable ID
DOUBLE RHVALS(LONS, LATS)
   ...
STATUS = NF_OPEN ('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_GET_VAR_DOUBLE (NCID, RHID, RHVALS)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.26 NF_GET_VARA_ *type*

The members of the NF_GET_VARA_ *type* family of functions read an array of values from a netCDF variable of an open netCDF dataset. The array is specified by giving a corner and a vector of edge lengths. The values are read into consecutive locations with the first dimension varying fastest. The netCDF dataset must be in data mode.

Usage

```
INTEGER FUNCTION NF_GET_VARA_TEXT(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  CHARACTER*(*) text)
INTEGER FUNCTION NF_GET_VARA_INT1(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  INTEGER*1 i1vals(*))
INTEGER FUNCTION NF_GET_VARA_INT2(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  INTEGER*2 i2vals(*))
INTEGER FUNCTION NF_GET_VARA_INT (INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  INTEGER ivals(*))
INTEGER FUNCTION NF_GET_VARA_REAL(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  REAL rvals(*))
INTEGER FUNCTION NF_GET_VARA_DOUBLE(INTEGER NCID, INTEGER VARID,
                                  INTEGER START(*), INTEGER COUNT(*),
                                  DOUBLE dvals(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`START` A vector of integers specifying the index in the variable where the first of the data values will be read. The indices are relative to 1, so for example, the first data value of a variable would have index (1, 1, ..., 1). The length of START must be the same as the number of dimensions of the specified variable. The elements of START correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the starting record number for reading the data values.

`COUNT` A vector of integers specifying the edge lengths along each dimension of the block of data values to be read. To read a single value, for example, specify COUNT as (1, 1, ..., 1). The length of COUNT is the number of dimensions of the specified variable. The elements of COUNT correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last element of COUNT corresponds to a count of the number of records to read.

Note: setting any element of the count array to zero causes the function to exit without error, and without doing anything.

`text`
`i1vals`
`i2vals`
`ivals`
`rvals`
`dvals` The block of data values to be read. The data should be of the type appropriate for the function called. You cannot read CHARACTER data from a numeric variable or numeric data from a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur (see Section "Type Conversion" in *The NetCDF Users Guide*).

Errors

NF_GET_VARA_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified corner indices were out of range for the rank of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension length will cause an error.
- The specified edge lengths added to the specified corner would have referenced data out of range for the rank of the specified variable. For example, an edge length that is larger than the corresponding dimension length minus the corner index will cause an error.
- One or more of the values are out of the range of values representable by the desired type.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_GET_VARA_DOUBLE to read all the values of the variable named rh from an existing netCDF dataset named foo.nc. For simplicity in this example, we assume that we know that rh is dimensioned with lon, lat, and time, and that there are ten lon values, five lat values, and three time values.

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIMS=3)                  ! number of dimensions
PARAMETER (TIMES=3, LATS=5, LONS=10) ! dimension lengths
INTEGER STATUS, NCID
INTEGER RHID                         ! variable ID
INTEGER START(NDIMS), COUNT(NDIMS)
DOUBLE RHVALS(LONS, LATS, TIMES)
DATA START /1, 1, 1/                 ! start at first value
DATA COUNT /LONS, LATS, TIMES/       ! get all the values
   ...
STATUS = NF_OPEN ('foo.nc', NF_NOWRITE, NCID)
```

```
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_GET_VARA_DOUBLE (NCID, RHID, START, COUNT, RHVALS)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.27 NF_GET_VARS_ *type*

The NF_GET_VARS_ *type* family of functions read a subsampled (strided) array section of values from a netCDF variable of an open netCDF dataset. The subsampled array section is specified by giving a corner, a vector of edge lengths, and a stride vector. The values are read with the first dimension of the netCDF variable varying fastest. The netCDF dataset must be in data mode.

Usage

```
INTEGER FUNCTION NF_GET_VARS_TEXT  (INTEGER NCID, INTEGER VARID,
                              INTEGER START(*), INTEGER COUNT(*),
                              INTEGER STRIDE(*),CHARACTER*(*) text)
INTEGER FUNCTION NF_GET_VARS_INT1  (INTEGER NCID, INTEGER VARID,
                              INTEGER START(*), INTEGER COUNT(*),
                              INTEGER STRIDE(*),INTEGER*1 i1vals(*))
INTEGER FUNCTION NF_GET_VARS_INT2  (INTEGER NCID, INTEGER VARID,
                              INTEGER START(*), INTEGER COUNT(*),
                              INTEGER STRIDE(*),INTEGER*2 i2vals(*))
INTEGER FUNCTION NF_GET_VARS_INT   (INTEGER NCID, INTEGER VARID,
                              INTEGER START(*), INTEGER COUNT(*),
                              INTEGER STRIDE(*), INTEGER ivals(*))
INTEGER FUNCTION NF_GET_VARS_REAL  (INTEGER NCID, INTEGER VARID,
                              INTEGER START(*), INTEGER COUNT(*),
                              INTEGER STRIDE(*), REAL rvals(*))
INTEGER FUNCTION NF_GET_VARS_DOUBLE(INTEGER NCID, INTEGER VARID,
                              INTEGER START(*), INTEGER COUNT(*),
                              INTEGER STRIDE(*), DOUBLE dvals(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`START` A vector of integers specifying the index in the variable from which the first of the data values will be read. The indices are relative to 1, so for example, the first data value of a variable would have index (1, 1, ..., 1). The elements of START correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the starting record number for reading the data values.

`COUNT` A vector of integers specifying the number of indices selected along each dimension. To read a single value, for example, specify COUNT as (1, 1, ..., 1). The elements of COUNT correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last element of COUNT corresponds to a count of the number of records to read.

Note: setting any element of the count array to zero causes the function to exit without error, and without doing anything.

`STRIDE` A vector of integers specifying, for each dimension, the interval between selected indices or the value 0. The elements of the vector correspond, in order, to the variable's dimensions. A value of 1 accesses adjacent values of the netCDF variable in the corresponding dimension; a value of 2 accesses every other value of the netCDF variable in the corresponding dimension; and so on. A 0 argument is treated as (1, 1, ..., 1).

`text`
`i1vals`
`i2vals`
`ivals`
`rvals`
`dvals` The block of data values to be read. The data should be of the type appropriate for the function called. You cannot read CHARACTER data from a numeric variable or numeric data from a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur (see Section "Type Conversion" in *The NetCDF Users Guide*).

Errors

NF_GET_VARS_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified start, count and stride generate an index which is out of range.
- One or more of the values are out of the range of values representable by the desired type.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_GET_VARS_DOUBLE to read every other value in each dimension of the variable named rh from an existing netCDF dataset named foo.nc. Values are assigned, using the same dimensional strides, to a 2-parameter array. For simplicity in this example, we assume that we know that rh is dimensioned with lon, lat, and time, and that there are ten lon values, five lat values, and three time values.

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIMS=3)                  ! number of dimensions
PARAMETER (TIMES=3, LATS=5, LONS=10) ! dimension lengths
INTEGER STATUS, NCID
INTEGER RHID ! variable ID
```

```
INTEGER START(NDIMS), COUNT(NDIMS), STRIDE(NDIMS)
DOUBLE DATA(LONS, LATS, TIMES)
DATA START /1, 1, 1/                ! start at first value
DATA COUNT /LONS, LATS, TIMES/
DATA STRIDE /2, 2, 2/
   ...
STATUS = NF_OPEN ('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_GET_VARS_DOUBLE(NCID,RHID,START,COUNT,STRIDE,DATA(1,1,1))
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.28 NF_GET_VARM_ *type*

The NF_GET_VARM_ *type* family of functions reads a mapped array section of values from a netCDF variable of an open netCDF dataset. The mapped array section is specified by giving a corner, a vector of edge lengths, a stride vector, and an index mapping vector. The index mapping vector is a vector of integers that specifies the mapping between the dimensions of a netCDF variable and the in-memory structure of the internal data array. No assumptions are made about the ordering or length of the dimensions of the data array. The netCDF dataset must be in data mode.

Usage

```
INTEGER FUNCTION NF_GET_VARM_TEXT  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            CHARACTER*(*) text)
INTEGER FUNCTION NF_GET_VARM_INT1  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            INTEGER*1 i1vals(*))
INTEGER FUNCTION NF_GET_VARM_INT2  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            INTEGER*2 i2vals(*))
INTEGER FUNCTION NF_GET_VARM_INT   (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            INTEGER ivals(*))
INTEGER FUNCTION NF_GET_VARM_REAL  (INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
                            INTEGER STRIDE(*), INTEGER IMAP(*),
                            REAL rvals(*))
INTEGER FUNCTION NF_GET_VARM_DOUBLE(INTEGER NCID, INTEGER VARID,
                            INTEGER START(*), INTEGER COUNT(*),
```

```
                              INTEGER STRIDE(*), INTEGER IMAP(*),
                              DOUBLE dvals(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`START` A vector of integers specifying the index in the variable from which the first of the data values will be read. The indices are relative to 1, so for example, the first data value of a variable would have index (1, 1, ..., 1). The elements of START correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the starting record number for reading the data values.

`COUNT` A vector of integers specifying the number of indices selected along each dimension. To read a single value, for example, specify COUNT as (1, 1, ..., 1). The elements of COUNT correspond, in order, to the variable's dimensions. Hence, if the variable is a record variable, the last element of COUNT corresponds to a count of the number of records to read.

Note: setting any element of the count array to zero causes the function to exit without error, and without doing anything.

`STRIDE` A vector of integers specifying, for each dimension, the interval between selected indices or the value 0. The elements of the vector correspond, in order, to the variable's dimensions. A value of 1 accesses adjacent values of the netCDF variable in the corresponding dimension; a value of 2 accesses every other value of the netCDF variable in the corresponding dimension; and so on. A 0 argument is treated as (1, 1, ..., 1).

`IMAP` A vector of integers that specifies the mapping between the dimensions of a netCDF variable and the in-memory structure of the internal data array. IMAP(1) gives the distance between elements of the internal array corresponding to the most rapidly varying dimension of the netCDF variable. IMAP(N) (where N is the rank of the netCDF variable) gives the distance between elements of the internal array corresponding to the most slowly varying dimension of the netCDF variable. Intervening IMAP elements correspond to other dimensions of the netCDF variable in the obvious way. Distances between elements are specified in units of elements (the distance between internal elements that occupy adjacent memory locations is 1 and not the element's byte-length as in netCDF 2).

`text`
`i1vals`
`i2vals`
`ivals`
`rvals`
`dvals` The block of data values to be read. The data should be of the type appropriate for the function called. You cannot read CHARACTER data from a numeric variable or numeric data from a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur (see Section "Type Conversion" in *The NetCDF Users Guide*).

Errors

NF_GET_VARM_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified START, COUNT, and STRIDE generate an index which is out of range. Note that no error checking is possible on the imap vector.
- One or more of the values are out of the range of values representable by the desired type.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

The following IMAP vector maps in the trivial way a 2x3x4 netCDF variable and an internal array of the same shape:

```
REAL A(2,3,4)       ! same shape as netCDF variable
INTEGER IMAP(3)
DATA IMAP /1, 2, 6/ ! netCDF dimension       inter-element distance
                    ! ----------------       ----------------------
                    ! most rapidly varying       1
                    ! intermediate               2 (=IMAP(1)*2)
                    ! most slowly varying        6 (=IMAP(2)*3)
```

Using the IMAP vector above with NF_GET_VARM_REAL obtains the same result as simply using NF_GET_VAR_REAL.

Here is an example of using NF_GET_VARM_REAL to transpose a netCDF variable named rh which is described by the FORTRAN declaration REAL RH(4,6) (note the size and order of the dimensions):

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIM=2)   ! rank of netCDF variable
INTEGER NCID         ! netCDF dataset ID
INTEGER STATUS       ! return code
INTEGER RHID         ! variable ID
INTEGER START(NDIM)  ! netCDF variable start point
INTEGER COUNT(NDIM)  ! size of internal array
INTEGER STRIDE(NDIM) ! netCDF variable subsampling intervals
INTEGER IMAP(NDIM)   ! internal array inter-element distances
REAL    RH(6,4)      ! note transposition of netCDF variable dimensions
DATA START   /1, 1/  ! start at first netCDF variable element
DATA COUNT   /4, 6/  ! entire netCDF variable; order corresponds
                     ! to netCDF variable -- not internal array
DATA STRIDE /1, 1/   ! sample every netCDF element
DATA IMAP   /6, 1/   ! would be /1, 4/ if not transposing
   ...
STATUS = NF_OPEN('foo.nc', NF_NOWRITE, NCID)
```

```
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID(NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_GET_VARM_REAL(NCID, RHID, START, COUNT, STRIDE, IMAP, RH)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

Here is another example of using NF_GET_VARM_REAL to simultaneously transpose and subsample the same netCDF variable, by accessing every other point of the netCDF variable:

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (NDIM=2)   ! rank of netCDF variable
INTEGER NCID         ! netCDF dataset ID
INTEGER STATUS       ! return code
INTEGER RHID         ! variable ID
INTEGER START(NDIM)  ! netCDF variable start point
INTEGER COUNT(NDIM)  ! size of internal array
INTEGER STRIDE(NDIM) ! netCDF variable subsampling intervals
INTEGER IMAP(NDIM)   ! internal array inter-element distances
REAL    RH(3,2)      ! note transposition of (subsampled) dimensions
DATA START   /1, 1/  ! start at first netCDF variable value
DATA COUNT   /2, 3/  ! order of (subsampled) dimensions corresponds
                     ! to netCDF variable -- not internal array
DATA STRIDE /2, 2/   ! sample every other netCDF element
DATA IMAP   /3, 1/   ! would be `1, 2' if not transposing
   ...
STATUS = NF_OPEN('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID(NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_GET_VARM_REAL(NCID, RHID, START, COUNT, STRIDE, IMAP, RH)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.29 Reading and Writing Character String Values

Character strings are not a primitive netCDF external data type, in part because FORTRAN does not support the abstraction of variable-length character strings (the FORTRAN LEN function returns the static length of a character string, not its dynamic length). As a result, a character string cannot be written or read as a single object in the netCDF interface. Instead, a character string must be treated as an array of characters, and array access must be used to read and write character strings as variable data in netCDF datasets. Furthermore, variable-length strings are not supported by the netCDF interface except by convention; for example, you may treat a zero byte as terminating a character string, but you must explicitly specify the length of strings to be read from and written to netCDF variables.

Character strings as attribute values are easier to use, since the strings are treated as a single unit for access. However, the value of a character-string attribute is still an array of characters with an explicit length that must be specified when the attribute is defined.

When you define a variable that will have character-string values, use a character-position dimension as the most quickly varying dimension for the variable (the first dimension for the variable in FORTRAN). The length of the character-position dimension will be the maximum string length of any value to be stored in the character-string variable. Space for maximum-length strings will be allocated in the disk representation of character-string variables whether you use the space or not. If two or more variables have the same maximum length, the same character-position dimension may be used in defining the variable shapes.

To write a character-string value into a character-string variable, use either entire variable access or array access. The latter requires that you specify both a corner and a vector of edge lengths. The character-position dimension at the corner should be one for FORTRAN. If the length of the string to be written is n, then the vector of edge lengths will specify n in the character-position dimension, and one for all the other dimensions:(n, 1, 1, ..., 1).

In FORTRAN, fixed-length strings may be written to a netCDF dataset without a terminating character, to save space. Variable-length strings should follow the C convention of writing strings with a terminating zero byte so that the intended length of the string can be determined when it is later read by either C or FORTRAN programs.

The FORTRAN interface for reading and writing strings requires the use of different functions for accessing string values and numeric values, because standard FORTRAN does not permit the same formal parameter to be used for both character values and numeric values. An additional argument, specifying the declared length of the character string passed as a value, is required for NF_PUT_VARA_TEXT and NF_GET_VARA_TEXT. The actual length of the string is specified as the value of the edge-length vector corresponding to the character-position dimension.

Here is an example that defines a record variable, tx, for character strings and stores a character-string value into the third record using NF_PUT_VARA_TEXT. In this example, we assume the string variable and data are to be added to an existing netCDF dataset named foo.nc that already has an unlimited record dimension time.

```
INCLUDE 'netcdf.inc'
   ...
INTEGER   TDIMS, TXLEN
PARAMETER (TDIMS=2)    ! number of TX dimensions
PARAMETER (TXLEN = 15) ! length of example string
INTEGER  NCID
INTEGER  CHID          ! char position dimension id
INTEGER  TIMEID        ! record dimension id
INTEGER  TXID          ! variable ID
INTEGER  TXDIMS(TDIMS) ! variable shape
INTEGER  TSTART(TDIMS), TCOUNT(TDIMS)
CHARACTER*40 TXVAL     ! max length 40
```

```
DATA TXVAL /'example string'/
   ...
TXVAL(TXLEN:TXLEN) = CHAR(0)   ! null terminate
   ...
STATUS = NF_OPEN('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_REDEF(NCID)        ! enter define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! define character-position dimension for strings of max length 40
STATUS = NF_DEF_DIM(NCID, "chid", 40, CHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! define a character-string variable
TXDIMS(1) = CHID   ! character-position dimension first
TXDIMS(2) = TIMEID
STATUS = NF_DEF_VAR(NCID, "tx", NF_CHAR, TDIMS, TXDIMS, TXID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_ENDDEF(NCID) ! leave define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! write txval into tx netCDF variable in record 3
TSTART(1) = 1      ! start at beginning of variable
TSTART(2) = 3      ! record number to write
TCOUNT(1) = TXLEN  ! number of chars to write
TCOUNT(2) = 1      ! only write one record
STATUS = NF_PUT_VARA_TEXT (NCID, TXID, TSTART, TCOUNT, TXVAL)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.30 Fill Values

What happens when you try to read a value that was never written in an open netCDF dataset? You might expect that this should always be an error, and that you should get an error message or an error status returned. You do get an error if you try to read data from a netCDF dataset that is not open for reading, if the variable ID is invalid for the specified netCDF dataset, or if the specified indices are not properly within the range defined by the dimension lengths of the specified variable. Otherwise, reading a value that was not written returns a special fill value used to fill in any undefined values when a netCDF variable is first written.

You may ignore fill values and use the entire range of a netCDF external data type, but in this case you should make sure you write all data values before reading them. If you know you will be writing all the data before reading it, you can specify that no prefilling of variables with fill values will occur by calling NF_SET_FILL before writing. This may provide a significant performance gain for netCDF writes.

The variable attribute _FillValue may be used to specify the fill value for a variable. Their are default fill values for each type, defined in the include file netcdf.inc: NF_FILL_CHAR, NF_FILL_INT1 (same as NF_FILL_BYTE), NF_FILL_INT2 (same as NF_FILL_SHORT), NF_FILL_INT, NF_FILL_REAL (same as NF_FILL_FLOAT), and NF_FILL_DOUBLE.

The netCDF byte and character types have different default fill values. The default fill value for characters is the zero byte, a useful value for detecting the end of variable-length C character strings. If you need a fill value for a byte variable, it is recommended that you explicitly define an appropriate _FillValue attribute, as generic utilities such as ncdump will not assume a default fill value for byte variables.

Type conversion for fill values is identical to type conversion for other values: attempting to convert a value from one type to another type that can't represent the value results in a range error. Such errors may occur on writing or reading values from a larger type (such as double) to a smaller type (such as float), if the fill value for the larger type cannot be represented in the smaller type.

6.31 NF_RENAME_VAR

The function NF_RENAME_VAR changes the name of a netCDF variable in an open netCDF dataset. If the new name is longer than the old name, the netCDF dataset must be in define mode. You cannot rename a variable to have the name of any existing variable.

Usage

```
INTEGER FUNCTION NF_RENAME_VAR (INTEGER NCID, INTEGER VARID,
                                CHARACTER*(*) NEWNAM)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID.

`NAME` New name for the specified variable.

Errors

NF_RENAME_VAR returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The new name is in use as the name of another variable.
- The variable ID is invalid for the specified netCDF dataset.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_RENAME_VAR to rename the variable rh to rel_hum in an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  STATUS, NCID
INTEGER  RHID             ! variable ID
   ...
STATUS = NF_OPEN ('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

```
   ...
STATUS = NF_REDEF (NCID)  ! enter definition mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_RENAME_VAR (NCID, RHID, 'rel_hum')
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_ENDDEF (NCID) ! leave definition mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

6.32 Change between Collective and Independent Parallel Access: NF_VAR_PAR_ACCESS

The function NF_VAR_PAR_ACCESS changes whether read/write operations on a parallel file system are performed collectively or independently (the default) on the variable. This function can only be called if the file was created with NF_CREATE_PAR (see Section 2.7 [NF_CREATE_PAR], page 12) or opened with NF_OPEN_PAR (see Section 2.10 [NF_OPEN_PAR], page 15).

This function is only available if the netCDF library was built with a HDF5 library for which –enable-parallel was used, and which was linked (like HDF5) to MPI libraries.

Calling this function affects only the open file - information about whether a variable is to be accessed collectively or independently is not written to the data file. Every time you open a file on a parallel file system, all variables default to independent operations. The change a variable to collective lasts only as long as that file is open.

The variable can be changed from collective to independent, and back, as often as desired.

Usage

```
INTEGER NF_VAR_PAR_ACCESS(INTEGER NCID, INTEGER VARID, INTEGER ACCESS);
```

`NCID` NetCDF ID, from a previous call to NF_OPEN_PAR (see Section 2.10 [NF_OPEN_PAR], page 15) or NF_CREATE_PAR (see Section 2.7 [NF_CREATE_PAR], page 12).

`varid` Variable ID.

`access` NF_INDEPENDENT to set this variable to independent operations. NF_COLLECTIVE to set it to collective operations.

Return Values

`NF_NOERR` No error.

`NF_ENOTVAR`
No variable found.

`NF_ENOTNC4`
Not a netCDF-4 file.

`NF_NOPAR` File not opened for parallel access.

Example

This example comes from test program nf_test/ftst_parallel.F. For this test to be run, netCDF must have been built with a parallel-enabled HDF5, and –enable-parallel-tests must have been used when configuring netcdf.

```
retval = nf_var_par_access(ncid, varid, nf_collective)
if (retval .ne. nf_noerr) stop 2
```

7 Attributes

7.1 Attributes Introduction

Attributes may be associated with each netCDF variable to specify such properties as units, special values, maximum and minimum valid values, scaling factors, and offsets. Attributes for a netCDF dataset are defined when the dataset is first created, while the netCDF dataset is in define mode. Additional attributes may be added later by reentering define mode. A netCDF attribute has a netCDF variable to which it is assigned, a name, a type, a length, and a sequence of one or more values. An attribute is designated by its variable ID and name. When an attribute name is not known, it may be designated by its variable ID and number in order to determine its name, using the function NF_INQ_ATTNAME.

The attributes associated with a variable are typically defined immediately after the variable is created, while still in define mode. The data type, length, and value of an attribute may be changed even when in data mode, as long as the changed attribute requires no more space than the attribute as originally defined.

It is also possible to have attributes that are not associated with any variable. These are called global attributes and are identified by using NF_GLOBAL as a variable pseudo-ID. Global attributes are usually related to the netCDF dataset as a whole and may be used for purposes such as providing a title or processing history for a netCDF dataset.

Attributes are much more useful when they follow established community conventions. See Section “Attribute Conventions” in *The NetCDF Users Guide*.

Operations supported on attributes are:

- Create an attribute, given its variable ID, name, data type, length, and value.
- Get attribute’s data type and length from its variable ID and name.
- Get attribute’s value from its variable ID and name.
- Copy attribute from one netCDF variable to another.
- Get name of attribute from its number.
- Rename an attribute.
- Delete an attribute.

7.2 NF_PUT_ATT_ *type*

The function NF_PUT_ATT_ *type* adds or changes a variable attribute or global attribute of an open netCDF dataset. If this attribute is new, or if the space required to store the attribute is greater than before, the netCDF dataset must be in define mode.

Usage

Although it’s possible to create attributes of all types, text and double attributes are adequate for most purposes.

```
INTEGER FUNCTION  NF_PUT_ATT_TEXT  (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME, INTEGER LEN,
                                    CHARACTER*(*) TEXT)
INTEGER FUNCTION  NF_PUT_ATT_INT1  (INTEGER NCID, INTEGER VARID,
```

```
                                    CHARACTER*(*) NAME, INTEGER XTYPE,
                                    LEN, INTEGER*1 I1VALS(*))
INTEGER FUNCTION  NF_PUT_ATT_INT2  (INTEGER NCID, INTEGER VARID,
                                     CHARACTER*(*) NAME, INTEGER XTYPE,
                                    LEN, INTEGER*2 I2VALS(*))
INTEGER FUNCTION  NF_PUT_ATT_INT   (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME, INTEGER XTYPE,
                                    LEN, INTEGER IVALS(*))
INTEGER FUNCTION  NF_PUT_ATT_REAL  (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME, INTEGER XTYPE,
                                    LEN, REAL RVALS(*))
INTEGER FUNCTION  NF_PUT_ATT_DOUBLE(INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME, INTEGER XTYPE,
                                    LEN, DOUBLE DVALS(*))
INTEGER FUNCTION  NF_PUT_ATT       (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME, INTEGER XTYPE,
                                    LEN, * VALS(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID of the variable to which the attribute will be assigned or NF_GLOBAL for a global attribute.

`NAME` Attribute name. Attribute name conventions are assumed by some netCDF generic applications, e.g., '`units`' as the name for a string attribute that gives the units for a netCDF variable. See Section "Attribute Conventions" in *The NetCDF Users Guide*.

`XTYPE` One of the set of predefined netCDF external data types. The type of this parameter, NF_TYPE, is defined in the netCDF header file. The valid netCDF external data types are NF_BYTE, NF_CHAR, NF_SHORT, NF_INT, NF_FLOAT, and NF_DOUBLE. Although it's possible to create attributes of all types, NF_CHAR and NF_DOUBLE attributes are adequate for most purposes.

`LEN` Number of values provided for the attribute.

`TEXT`
`I1VALS`
`I2VALS`
`IVALS`
`RVALS`
`DVALS`
`VALS` An array of LEN attribute values. The data should be of a type appropriate for the function called. You cannot write CHARACTER data into a numeric attribute or numeric data into a text attribute. For numeric data, if the type of data differs from the attribute type, type conversion will occur See Section "Type Conversion" in *The NetCDF Users Guide*.

Errors

NF_PUT_ATT_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified netCDF type is invalid.
- The specified length is negative.
- The specified open netCDF dataset is in data mode and the specified attribute would expand.
- The specified open netCDF dataset is in data mode and the specified attribute does not already exist.
- The specified netCDF ID does not refer to an open netCDF dataset.
- The number of attributes for this variable exceeds NF_MAX_ATTRS.

Example

Here is an example using NF_PUT_ATT_DOUBLE to add a variable attribute named valid_range for a netCDF variable named rh and a global attribute named title to an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID
INTEGER RHID                 ! variable ID
DOUBLE RHRNGE(2)
DATA RHRNGE /0.0D0, 100.0D0/
   ...
STATUS = NF_OPEN ('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_REDEF (NCID)     ! enter define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_PUT_ATT_DOUBLE (NCID, RHID, 'valid_range', NF_DOUBLE, &
                            2, RHRNGE)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_PUT_ATT_TEXT (NCID, NF_GLOBAL, 'title', 19,
                          'example netCDF dataset')
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_ENDDEF (NCID)    ! leave define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

7.3 NF_INQ_ATT Family

This family of functions returns information about a netCDF attribute. All but one of these functions require the variable ID and attribute name; the exception is NF_INQ_ATTNAME. Information about an attribute includes its type, length, name, and number. See the NF_GET_ATT family for getting attribute values.

The function NF_INQ_ATTNAME gets the name of an attribute, given its variable ID and number. This function is useful in generic applications that need to get the names of all the attributes associated with a variable, since attributes are accessed by name rather than number in all other attribute functions. The number of an attribute is more volatile than the name, since it can change when other attributes of the same variable are deleted. This is why an attribute number is not called an attribute ID.

The function NF_INQ_ATT returns the attribute's type and length. The other functions each return just one item of information about an attribute.

Usage

```
INTEGER FUNCTION NF_INQ_ATT    (INTEGER NCID, INTEGER VARID,
                                CHARACTER*(*) NAME, INTEGER xtype,
                                INTEGER len)
INTEGER FUNCTION NF_INQ_ATTTYPE(INTEGER NCID, INTEGER VARID,
                                CHARACTER*(*) NAME, INTEGER xtype)
INTEGER FUNCTION NF_INQ_ATTLEN (INTEGER NCID, INTEGER VARID,
                                CHARACTER*(*) NAME, INTEGER len)
INTEGER FUNCTION NF_INQ_ATTNAME(INTEGER NCID, INTEGER VARID,
                                INTEGER ATTNUM, CHARACTER*(*) name)
INTEGER FUNCTION NF_INQ_ATTID  (INTEGER NCID, INTEGER VARID,
                                CHARACTER*(*) NAME, INTEGER attnum)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID of the attribute's variable, or NF_GLOBAL for a global attribute.

`NAME` Attribute name. For NF_INQ_ATTNAME, this is a pointer to the location for the returned attribute name.

`xtype` Returned attribute type, one of the set of predefined netCDF external data types. The valid netCDF external data types are NF_BYTE, NF_CHAR, NF_SHORT, NF_INT, NF_FLOAT, and NF_DOUBLE.

`len` Returned number of values currently stored in the attribute. For a string-valued attribute, this is the number of characters in the string.

`attnum` For NF_INQ_ATTNAME, the input attribute number; for NF_INQ_ATTID, the returned attribute number. The attributes for each variable are numbered from 1 (the first attribute) to NATTS, where NATTS is the number of attributes for the variable, as returned from a call to NF_INQ_VARNATTS.

(If you already know an attribute name, knowing its number is not very useful, because accessing information about an attribute requires its name.)

Errors

Each function returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF dataset.
- For NF_INQ_ATTNAME, the specified attribute number is negative or more than the number of attributes defined for the specified variable.

Example

Here is an example using NF_INQ_ATT to find out the type and length of a variable attribute named valid_range for a netCDF variable named rh and a global attribute named title in an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS, NCID
INTEGER RHID               ! variable ID
INTEGER VRLEN, TLEN        ! attribute lengths
   ...
STATUS = NF_OPEN ('foo.nc', NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_ATTLEN (NCID, RHID, 'valid_range', VRLEN)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_INQ_ATTLEN (NCID, NF_GLOBAL, 'title', TLEN)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

7.4 NF_GET_ATT_ *type*

Members of the NF_GET_ATT_ *type* family of functions get the value(s) of a netCDF attribute, given its variable ID and name.

Usage

```
INTEGER FUNCTION NF_GET_ATT_TEXT   (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME,
                                    CHARACTER*(*) text)
INTEGER FUNCTION NF_GET_ATT_INT1   (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME,
                                    INTEGER*1 i1vals(*))
INTEGER FUNCTION NF_GET_ATT_INT2   (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME,
                                    INTEGER*2 i2vals(*))
```

```
INTEGER FUNCTION NF_GET_ATT_INT    (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME,
                                    INTEGER ivals(*))
INTEGER FUNCTION NF_GET_ATT_REAL   (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME,
                                    REAL rvals(*))
INTEGER FUNCTION NF_GET_ATT_DOUBLE (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME,
                                    DOUBLE dvals(*))
INTEGER FUNCTION NF_GET_ATT        (INTEGER NCID, INTEGER VARID,
                                    CHARACTER*(*) NAME, * vals(*))
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` Variable ID of the attribute's variable, or NF_GLOBAL for a global attribute.

`NAME` Attribute name.

`TEXT`
`I1VALS`
`I2VALS`
`IVALS`
`RVALS`
`DVALS`
`VALS` Returned attribute values. All elements of the vector of attribute values are returned, so you must provide enough space to hold them. If you don't know how much space to reserve, call NF_INQ_ATTLEN first to find out the length of the attribute. You cannot read character data from a numeric variable or numeric data from a text variable. For numeric data, if the type of data differs from the netCDF variable type, type conversion will occur. See Section "Type Conversion" in *The The NetCDF Users Guide*.

Errors

NF_GET_ATT_ *type* returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF dataset.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF dataset.
- One or more of the attribute values are out of the range of values representable by the desired type.

Example

Here is an example using NF_GET_ATT_DOUBLE to determine the values of a variable attribute named valid_range for a netCDF variable named rh and a global attribute named title in an existing netCDF dataset named foo.nc. In this example, it is assumed that we don't know how many values will be returned, but that we do know the types of the attributes. Hence, to allocate enough space to store them, we must first inquire about the length of the attributes.

```
INCLUDE 'netcdf.inc'
   ...
PARAMETER (MVRLEN=3)           ! max number of "valid_range" values
PARAMETER (MTLEN=80)           ! max length of "title" attribute
INTEGER STATUS, NCID
INTEGER RHID                   ! variable ID
INTEGER VRLEN, TLEN            ! attribute lengths
DOUBLE PRECISION VRVAL(MVRLEN) ! vr attribute values
CHARACTER*80 TITLE             ! title attribute values
   ...
STATUS = NF_OPEN ('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! find out attribute lengths, to make sure we have enough space
STATUS = NF_INQ_ATTLEN (NCID, RHID, 'valid_range', VRLEN)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_INQ_ATTLEN (NCID, NF_GLOBAL, 'title', TLEN)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
! get attribute values, if not too big
IF (VRLEN .GT. MVRLEN) THEN
    WRITE (*,*) 'valid_range attribute too big!'
    CALL EXIT
ELSE
    STATUS = NF_GET_ATT_DOUBLE (NCID, RHID, 'valid_range', VRVAL)
    IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
ENDIF
IF (TLEN .GT. MTLEN) THEN
    WRITE (*,*) 'title attribute too big!'
    CALL EXIT
ELSE
    STATUS = NF_GET_ATT_TEXT (NCID, NF_GLOBAL, 'title', TITLE)
    IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
ENDIF
```

7.5 NF_COPY_ATT

The function NF_COPY_ATT copies an attribute from one open netCDF dataset to another. It can also be used to copy an attribute from one variable to another within the same netCDF.

If used to copy an attribute of user-defined type, then that user-defined type must already be defined in the target file. In the case of user-defined attributes, enddef/redef is called for ncid_in and ncid_out if they are in define mode. (This is the ensure that all user-defined types are committed to the file(s) before the copy is attempted.)

Usage

```
INTEGER FUNCTION NF_COPY_ATT (INTEGER NCID_IN, INTEGER VARID_IN,
                              CHARACTER*(*) NAME, INTEGER NCID_OUT,
                              INTEGER VARID_OUT)
```

`NCID_IN` The netCDF ID of an input netCDF dataset from which the attribute will be copied, from a previous call to NF_OPEN or NF_CREATE.

`VARID_IN` ID of the variable in the input netCDF dataset from which the attribute will be copied, or NF_GLOBAL for a global attribute.

`NAME` Name of the attribute in the input netCDF dataset to be copied.

`NCID_OUT` The netCDF ID of the output netCDF dataset to which the attribute will be copied, from a previous call to NF_OPEN or NF_CREATE. It is permissible for the input and output netCDF IDs to be the same. The output netCDF dataset should be in define mode if the attribute to be copied does not already exist for the target variable, or if it would cause an existing target attribute to grow.

`VARID_OUT`
ID of the variable in the output netCDF dataset to which the attribute will be copied, or NF_GLOBAL to copy to a global attribute.

Errors

NF_COPY_ATT returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The input or output variable ID is invalid for the specified netCDF dataset.
- The specified attribute does not exist.
- The output netCDF is not in define mode and the attribute is new for the output dataset is larger than the existing attribute.
- The input or output netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_COPY_ATT to copy the variable attribute units from the variable rh in an existing netCDF dataset named foo.nc to the variable avgrh in another existing netCDF dataset named bar.nc, assuming that the variable avgrh already exists, but does not yet have a units attribute:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS             ! error status
INTEGER NCID1, NCID2       ! netCDF IDs
INTEGER RHID, AVRHID       ! variable IDs
   ...
STATUS = NF_OPEN ('foo.nc', NF_NOWRITE, NCID1)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_OPEN ('bar.nc', NF_WRITE, NCID2)
```

```
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID1, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_INQ_VARID (NCID2, 'avgrh', AVRHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_REDEF (NCID2)  ! enter define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
! copy variable attribute from "rh" to "avgrh"
STATUS = NF_COPY_ATT (NCID1, RHID, 'units', NCID2, AVRHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_ENDDEF (NCID2) ! leave define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

7.6 NF_RENAME_ATT

The function NF_RENAME_ATT changes the name of an attribute. If the new name is longer than the original name, the netCDF dataset must be in define mode. You cannot rename an attribute to have the same name as another attribute of the same variable.

Usage

```
INTEGER FUNCTION NF_RENAME_ATT (INTEGER NCID, INTEGER VARID,
                                CHARACTER*(*) NAME,
                                CHARACTER*(*) NEWNAME)
```

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE

`VARID` ID of the attribute's variable, or NF_GLOBAL for a global attribute

`NAME` The current attribute name.

`NEWNAME` The new name to be assigned to the specified attribute. If the new name is longer than the current name, the netCDF dataset must be in define mode.

Errors

NF_RENAME_ATT returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified variable ID is not valid.
- The new attribute name is already in use for another attribute of the specified variable.
- The specified netCDF dataset is in data mode and the new name is longer than the old name.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_RENAME_ATT to rename the variable attribute units to Units for a variable rh in an existing netCDF dataset named foo.nc:

```
INCLUDE "netcdf.inc"
   ...
INTEGER STATUS   ! error status
INTEGER NCID     ! netCDF ID
INTEGER RHID     ! variable ID
   ...
STATUS = NF_OPEN ("foo.nc", NF_NOWRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, "rh", RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! rename attribute
STATUS = NF_RENAME_ATT (NCID, RHID, "units", "Units")
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

7.7 NF_DEL_ATT

The function NF_DEL_ATT deletes a netCDF attribute from an open netCDF dataset. The netCDF dataset must be in define mode.

Usage

INTEGER FUNCTION NF_DEL_ATT (INTEGER NCID, INTEGER VARID, CHARACTER*(*) NAME)

`NCID` NetCDF ID, from a previous call to NF_OPEN or NF_CREATE.

`VARID` ID of the attribute's variable, or NF_GLOBAL for a global attribute.

`NAME` The name of the attribute to be deleted.

Errors

NF_DEL_ATT returns the value NF_NOERR if no errors occurred. Otherwise, the returned status indicates an error. Possible causes of errors include:

- The specified variable ID is not valid.
- The specified netCDF dataset is in data mode.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF dataset.

Example

Here is an example using NF_DEL_ATT to delete the variable attribute Units for a variable rh in an existing netCDF dataset named foo.nc:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER STATUS            ! error status
INTEGER NCID              ! netCDF ID
INTEGER RHID              ! variable ID
   ...
STATUS = NF_OPEN ('foo.nc', NF_WRITE, NCID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
STATUS = NF_INQ_VARID (NCID, 'rh', RHID)
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
   ...
! delete attribute
STATUS = NF_REDEF (NCID)  ! enter define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_DEL_ATT (NCID, RHID, 'Units')
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
STATUS = NF_ENDDEF (NCID) ! leave define mode
IF (STATUS .NE. NF_NOERR) CALL HANDLE_ERR(STATUS)
```

Appendix A NetCDF 2 to NetCDF 3 Fortran 77 Transition Guide

A.1 Overview of FORTRAN interface changes

NetCDF version 3 includes a complete rewrite of the netCDF library. It is about twice as fast as the previous version. The netCDF file format is unchanged, so files written with version 3 can be read with version 2 code and vice versa.

The core library is now written in ANSI C. You must have an ANSI C compiler to compile this version. The FORTRAN interface is layered on top of the C interface using a different technique than was used in netCDF-2.

Rewriting the library offered an opportunity to implement improved C and FORTRAN interfaces that provide some significant benefits:

- type safety, by eliminating the need to use type punning in arguments;
- automatic type conversions, by eliminating the undesirable coupling between the language-independent external netCDF types (NF_BYTE, ..., NF_DOUBLE) and language-dependent internal data types (INT*1, ..., DOUBLE PRECISION);
- support for future enhancements, by eliminating obstacles to the clean addition of support for packed data and multithreading;
- more standard error behavior, by uniformly communicating an error status back to the calling program in the return value of each function.

It is not necessary to rewrite programs that use the version 2 FORTRAN interface, because the netCDF-3 library includes a backward compatibility interface that supports all the old functions, globals, and behavior. We are hoping that the benefits of the new interface will be an incentive to use it in new netCDF applications. It is possible to convert old applications to the new interface incrementally, replacing netCDF-2 calls with the corresponding netCDF-3 calls one at a time.

Other changes in the implementation of netCDF result in improved portability, maintainability, and performance on most platforms. A clean separation between I/O and type layers facilitates platform-specific optimizations. The new library no longer uses a vendor-provided XDR library, which simplifies linking programs that use netCDF and speeds up data access significantly in most cases.

A.2 The New FORTRAN Interface

First, here's an example of FORTRAN code that uses the netCDF-2 interface:

```
! Use a buffer big enough for values of any type
DOUBLE PRECISION DBUF(NDATA)
REAL RBUF(NDATA)
...
EQUIVALENCE (RBUF, DBUF), ...
INT XTYPE     ! to hold the actual type of the data
INT STATUS    ! for error status
! Get the actual data type
CALL NCVINQ(NCID, VARID, ...,XTYPE, ...)
...
```

```
! Get the data
CALL NCVGT(NCID, VARID, START, COUNT, DBUF, STATUS)
IF(STATUS .NE. NCNOERR) THEN
   PRINT *, 'Cannot get data, error code =', STATUS
   ! Deal with error
   ...
ENDIF
IF (XTYPE .EQ. NCDOUBLE) THEN
   CALL DANALYZE(DBUF)
ELSEIF (XTYPE .EQ. NCFLOAT) THEN
   CALL RANALYZE(RBUF)
...
ENDIF
```

Here's how you might handle this with the new netCDF-3 FORTRAN interface:

```
! I want to use doubles for my analysis
DOUBLE PRECISION DBUF(NDATA)
INT STATUS
! So I use a function that gets the data as doubles.
STATUS = NF_GET_VARA_DOUBLE(NCID, VARID, START, COUNT, DBUF)
IF(STATUS .NE. NF_NOERR) THEN
   PRINT *, 'Cannot get data, ', NF_STRERROR(STATUS)
   ! Deal with error
   ...
ENDIF
CALL DANALYZE(DBUF)
```

The example above illustrates changes in function names, data type conversion, and error handling, discussed in detail in the sections below.

A.3 Function Naming Conventions

The netCDF-3 Fortran 77 library employs a naming convention intended to make netCDF programs more readable. For example, the name of the function to rename a variable is now NF_RENAME_VAR instead of the previous NCVREN.

All netCDF-3 FORTRAN function names begin with the NF_ prefix. The second part of the name is a verb, like GET, PUT, INQ (for inquire), or OPEN. The third part of the name is typically the object of the verb: for example DIM, VAR, or ATT for functions dealing with dimensions, variables, or attributes. To distinguish the various I/O operations for variables, a single character modifier is appended to VAR:

- VAR entire variable access
- VAR1 single value access
- VARA array or array section access
- VARS strided access to a subsample of values
- VARM mapped access to values not contiguous in memory

At the end of the name for variable and attribute functions, there is a component indicating the type of the final argument: TEXT, INT1, INT2, INT, REAL, or DOUBLE.

This part of the function name indicates the type of the data container you are using in your program: character string, 1-byte integer, and so on.

Also, all PARAMETER names in the public FORTRAN interface begin with the prefix NF_. For example, the PARAMETER which was formerly MAXNCNAM is now NF_MAX_NAME, and the former FILFLOAT is now NF_FILL_FLOAT.

As previously mentioned, all the old names are still supported for backward compatibility.

A.4 Type Conversion

With the new interface, users need not be aware of the external type of numeric variables, since automatic conversion to or from any desired numeric type is now available. You can use this feature to simplify code, by making it independent of external types. The elimination of type punning prevents some kinds of type errors that could occur with the previous interface. Programs may be made more robust with the new interface, because they need not be changed to accommodate a change to the external type of a variable.

If conversion to or from an external numeric type is necessary, it is handled by the library. This automatic conversion and separation of external data representation from internal data types will become even more important in netCDF version 4, when new external types will be added for packed data for which there is no natural corresponding internal type, for example, arrays of 11-bit values.

Converting from one numeric type to another may result in an error if the target type is not capable of representing the converted value. (In netCDF-2, such overflows can only happen in the XDR layer.) For example, a REAL may not be able to hold data stored externally as an NF_DOUBLE (an IEEE floating-point number). When accessing an array of values, an NF_ERANGE error is returned if one or more values are out of the range of representable values, but other values are converted properly.

Note that mere loss of precision in type conversion does not return an error. Thus, if you read double precision values into an INTEGER, for example, no error results unless the magnitude of the double precision value exceeds the representable range of INTEGERs on your platform. Similarly, if you read a large integer into a REAL incapable of representing all the bits of the integer in its mantissa, this loss There are two new functions in netCDF-3 that don't correspond to any netCDF-2 functions: NF_INQ_LIBVERS and NF_STRERROR. The version ation The previous implementation returned an error when the same dimension was used more than once in specifying the shape of a variable in ncvardef. This restriction is relaxed in the netCDF-3 implementation, because an autocorrelation matrix is a good example where using the same dimension twice makes sense.

In the new interface, units for the IMAP argument to the NF_PUT_VARM and NF_GET_VARM families of functions are now in terms of the number of data elements of the desired internal type, not in terms of bytes as in the netCDF version-2 mapped access interfaces.

Following is a table of netCDF-2 function names and names of the corresponding netCDF-3 functions. For parameter lists of netCDF-2 functions, see the netCDF-2 User's Guide.

`NCABOR` NF_ABORT

`NCACPY` NF_COPY_ATT

`NCADEL` NF_DEL_ATT

`NCAGT` NF_GET_ATT_DOUBLE, NF_GET_ATT_REAL, NF_GET_ATT_INT, NF_GET_ATT_INT1, NF_GET_ATT_INT2

`NCAGTC` NF_GET_ATT_TEXT

`NCAINQ` NF_INQ_ATT, NF_INQ_ATTID, NF_INQ_ATTLEN, NF_INQ_ATTTYPE

`NCANAM` NF_INQ_ATTNAME

`NCAPT` NF_PUT_ATT_DOUBLE, NF_PUT_ATT_REAL, NF_PUT_ATT_INT, NF_PUT_ATT_INT1NF_PUT

Appendix B Summary of FORTRAN 77 Interface

Input parameters are in upper case, output parameters are in lower case. The FORTRAN types of all the parameters are listed alphabetically by parameter name below the function declarations.

```
CHARACTER*80 FUNCTION  NF_INQ_LIBVERS()
CHARACTER*80 FUNCTION  NF_STRERROR  (NCERR)
INTEGER FUNCTION  NF_CREATE        (PATH, CMODE, ncid)
INTEGER FUNCTION  NF_OPEN          (PATH, MODE, ncid)
INTEGER FUNCTION  NF_SET_FILL      (NCID, FILLMODE, old_mode)
INTEGER FUNCTION  NF_REDEF         (NCID)
INTEGER FUNCTION  NF_ENDDEF        (NCID)
INTEGER FUNCTION  NF_SYNC          (NCID)
INTEGER FUNCTION  NF_ABORT         (NCID)
INTEGER FUNCTION  NF_CLOSE         (NCID)
INTEGER FUNCTION  NF_INQ           (NCID, ndims, nvars, ngatts,
                                    unlimdimid)
INTEGER FUNCTION  NF_INQ_NDIMS     (NCID, ndims)
INTEGER FUNCTION  NF_INQ_NVARS     (NCID, nvars)
INTEGER FUNCTION  NF_INQ_NATTS     (NCID, ngatts)
INTEGER FUNCTION  NF_INQ_UNLIMDIM  (NCID, unlimdimid)
INTEGER FUNCTION  NF_DEF_DIM       (NCID, NAME, LEN, dimid)
INTEGER FUNCTION  NF_INQ_DIMID     (NCID, NAME, dimid)
INTEGER FUNCTION  NF_INQ_DIM       (NCID, DIMID, name, len)
INTEGER FUNCTION  NF_INQ_DIMNAME   (NCID, DIMID, name)
INTEGER FUNCTION  NF_INQ_DIMLEN    (NCID, DIMID, len)
INTEGER FUNCTION  NF_RENAME_DIM    (NCID, DIMID, NAME)

INTEGER FUNCTION  NF_DEF_VAR       (NCID, NAME, XTYPE, NDIMS, DIMIDS,
                                    varid)
INTEGER FUNCTION  NF_INQ_VAR       (NCID, VARID, name, xtype, ndims,
                                    dimids, natts)
INTEGER FUNCTION  NF_INQ_VARID     (NCID, NAME, varid)
INTEGER FUNCTION  NF_INQ_VARNAME   (NCID, VARID, name)
INTEGER FUNCTION  NF_INQ_VARTYPE   (NCID, VARID, xtype)
INTEGER FUNCTION  NF_INQ_VARNDIMS  (NCID, VARID, ndims)
INTEGER FUNCTION  NF_INQ_VARDIMID  (NCID, VARID, DIMIDS)
INTEGER FUNCTION  NF_INQ_VARNATTS  (NCID, VARID, natts)
INTEGER FUNCTION  NF_RENAME_VAR    (NCID, VARID, NAME)
INTEGER FUNCTION  NF_PUT_VAR_TEXT  (NCID, VARID, TEXT)
INTEGER FUNCTION  NF_GET_VAR_TEXT  (NCID, VARID, text)
INTEGER FUNCTION  NF_PUT_VAR_INT1  (NCID, VARID, I1VAL)
INTEGER FUNCTION  NF_GET_VAR_INT1  (NCID, VARID, i1val)
INTEGER FUNCTION  NF_PUT_VAR_INT2  (NCID, VARID, I2VAL)
INTEGER FUNCTION  NF_GET_VAR_INT2  (NCID, VARID, i2val)
INTEGER FUNCTION  NF_PUT_VAR_INT   (NCID, VARID, IVAL)
INTEGER FUNCTION  NF_GET_VAR_INT   (NCID, VARID, ival)
```

```
INTEGER FUNCTION  NF_PUT_VAR_REAL   (NCID, VARID, RVAL)
INTEGER FUNCTION  NF_GET_VAR_REAL   (NCID, VARID, rval)
INTEGER FUNCTION  NF_PUT_VAR_DOUBLE (NCID, VARID, DVAL)
INTEGER FUNCTION  NF_GET_VAR_DOUBLE (NCID, VARID, dval)
INTEGER FUNCTION  NF_PUT_VAR1_TEXT  (NCID, VARID, INDEX, TEXT)
INTEGER FUNCTION  NF_GET_VAR1_TEXT  (NCID, VARID, INDEX, text)
INTEGER FUNCTION  NF_PUT_VAR1_INT1  (NCID, VARID, INDEX, I1VAL)
INTEGER FUNCTION  NF_GET_VAR1_INT1  (NCID, VARID, INDEX, i1val)
INTEGER FUNCTION  NF_PUT_VAR1_INT2  (NCID, VARID, INDEX, I2VAL)
INTEGER FUNCTION  NF_GET_VAR1_INT2  (NCID, VARID, INDEX, i2val)
INTEGER FUNCTION  NF_PUT_VAR1_INT   (NCID, VARID, INDEX, IVAL)
INTEGER FUNCTION  NF_GET_VAR1_INT   (NCID, VARID, INDEX, ival)
INTEGER FUNCTION  NF_PUT_VAR1_REAL  (NCID, VARID, INDEX, RVAL)
INTEGER FUNCTION  NF_GET_VAR1_REAL  (NCID, VARID, INDEX, rval)
INTEGER FUNCTION  NF_PUT_VAR1_DOUBLE(NCID, VARID, INDEX, DVAL)
INTEGER FUNCTION  NF_GET_VAR1_DOUBLE(NCID, VARID, INDEX, dval)
INTEGER FUNCTION  NF_PUT_VARA_TEXT  (NCID, VARID, START, COUNT, TEXT)
INTEGER FUNCTION  NF_GET_VARA_TEXT  (NCID, VARID, START, COUNT, text)
INTEGER FUNCTION  NF_PUT_VARA_INT1  (NCID, VARID, START, COUNT, I1VALS)
INTEGER FUNCTION  NF_GET_VARA_INT1  (NCID, VARID, START, COUNT, i1vals)
INTEGER FUNCTION  NF_PUT_VARA_INT2  (NCID, VARID, START, COUNT, I2VALS)
INTEGER FUNCTION  NF_GET_VARA_INT2  (NCID, VARID, START, COUNT, i2vals)
INTEGER FUNCTION  NF_PUT_VARA_INT   (NCID, VARID, START, COUNT, IVALS)
INTEGER FUNCTION  NF_GET_VARA_INT   (NCID, VARID, START, COUNT, ivals)
INTEGER FUNCTION  NF_PUT_VARA_REAL  (NCID, VARID, START, COUNT, RVALS)
INTEGER FUNCTION  NF_GET_VARA_REAL  (NCID, VARID, START, COUNT, rvals)
INTEGER FUNCTION  NF_PUT_VARA_DOUBLE(NCID, VARID, START, COUNT, DVALS)
INTEGER FUNCTION  NF_GET_VARA_DOUBLE(NCID, VARID, START, COUNT, dvals)
INTEGER FUNCTION  NF_PUT_VARS_TEXT  (NCID, VARID, START, COUNT, STRIDE,
                                     TEXT)
INTEGER FUNCTION  NF_GET_VARS_TEXT  (NCID, VARID, START, COUNT, STRIDE,
                                     text)
INTEGER FUNCTION  NF_PUT_VARS_INT1  (NCID, VARID, START, COUNT, STRIDE,
                                     I1VALS)
INTEGER FUNCTION  NF_GET_VARS_INT1  (NCID, VARID, START, COUNT, STRIDE,
                                     i1vals)
INTEGER FUNCTION  NF_PUT_VARS_INT2  (NCID, VARID, START, COUNT, STRIDE,
                                     I2VALS)
INTEGER FUNCTION  NF_GET_VARS_INT2  (NCID, VARID, START, COUNT, STRIDE,
                                     i2vals)
INTEGER FUNCTION  NF_PUT_VARS_INT   (NCID, VARID, START, COUNT, STRIDE,
                                     IVALS)
INTEGER FUNCTION  NF_GET_VARS_INT   (NCID, VARID, START, COUNT, STRIDE,
                                     ivals)
INTEGER FUNCTION  NF_PUT_VARS_REAL  (NCID, VARID, START, COUNT, STRIDE,
                                     RVALS)
INTEGER FUNCTION  NF_GET_VARS_REAL  (NCID, VARID, START, COUNT, STRIDE,
```

```
                                    rvals)
INTEGER FUNCTION  NF_PUT_VARS_DOUBLE(NCID, VARID, START, COUNT, STRIDE,
                                    DVALS)
INTEGER FUNCTION  NF_GET_VARS_DOUBLE(NCID, VARID, START, COUNT, STRIDE,
                                    dvals)
INTEGER FUNCTION  NF_PUT_VARM_TEXT  (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, TEXT)
INTEGER FUNCTION  NF_GET_VARM_TEXT  (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, text)
INTEGER FUNCTION  NF_PUT_VARM_INT1  (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, I1VALS)
INTEGER FUNCTION  NF_GET_VARM_INT1  (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, i1vals)
INTEGER FUNCTION  NF_PUT_VARM_INT2  (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, I2VALS)
INTEGER FUNCTION  NF_GET_VARM_INT2  (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, i2vals)
INTEGER FUNCTION  NF_PUT_VARM_INT   (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, IVALS)
INTEGER FUNCTION  NF_GET_VARM_INT   (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, ivals)
INTEGER FUNCTION  NF_PUT_VARM_REAL  (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, RVALS)
INTEGER FUNCTION  NF_GET_VARM_REAL  (NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, rvals)
INTEGER FUNCTION  NF_PUT_VARM_DOUBLE(NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, DVALS)
INTEGER FUNCTION  NF_GET_VARM_DOUBLE(NCID, VARID, START, COUNT, STRIDE,
                                    IMAP, dvals)

INTEGER FUNCTION  NF_INQ_ATT        (NCID, VARID, NAME, xtype, len)
INTEGER FUNCTION  NF_INQ_ATTID      (NCID, VARID, NAME, attnum)
INTEGER FUNCTION  NF_INQ_ATTTYPE    (NCID, VARID, NAME, xtype)
INTEGER FUNCTION  NF_INQ_ATTLEN     (NCID, VARID, NAME, len)
INTEGER FUNCTION  NF_INQ_ATTNAME    (NCID, VARID, ATTNUM, name)
INTEGER FUNCTION  NF_COPY_ATT       (NCID_IN, VARID_IN, NAME,
                                    NCID_OUT, VARID_OUT)
INTEGER FUNCTION  NF_RENAME_ATT     (NCID, VARID, CURNAME, NEWNAME)
INTEGER FUNCTION  NF_DEL_ATT        (NCID, VARID, NAME)
INTEGER FUNCTION  NF_PUT_ATT_TEXT   (NCID, VARID, NAME, LEN, TEXT)
INTEGER FUNCTION  NF_GET_ATT_TEXT   (NCID, VARID, NAME, text)
INTEGER FUNCTION  NF_PUT_ATT_INT1   (NCID, VARID, NAME, XTYPE, LEN,
                                    I1VALS)
INTEGER FUNCTION  NF_GET_ATT_INT1   (NCID, VARID, NAME, i1vals)
INTEGER FUNCTION  NF_PUT_ATT_INT2   (NCID, VARID, NAME, XTYPE, LEN,
                                    I2VALS)
INTEGER FUNCTION  NF_GET_ATT_INT2   (NCID, VARID, NAME, i2vals)
```

```
INTEGER FUNCTION  NF_PUT_ATT_INT    (NCID, VARID, NAME, XTYPE, LEN,
                                     IVALS)
INTEGER FUNCTION  NF_GET_ATT_INT    (NCID, VARID, NAME, ivals)
INTEGER FUNCTION  NF_PUT_ATT_REAL   (NCID, VARID, NAME, XTYPE, LEN,
                                     RVALS)
INTEGER FUNCTION  NF_GET_ATT_REAL   (NCID, VARID, NAME, rvals)
INTEGER FUNCTION  NF_PUT_ATT_DOUBLE (NCID, VARID, NAME, XTYPE, LEN,
                                     DVALS)
INTEGER FUNCTION  NF_GET_ATT_DOUBLE (NCID, VARID, NAME, dvals)

INTEGER       ATTNUM       ! attribute number
INTEGER       attnum       ! returned attribute number
INTEGER       CMODE        ! NF_NOCLOBBER, NF_SHARE flags expression
INTEGER       COUNT        ! array of edge lengths of block of values
CHARACTER(*)  CURNAME      ! current name (before renaming)
INTEGER       DIMID        ! dimension ID
INTEGER       dimid        ! returned dimension ID
INTEGER       DIMIDS       ! list of dimension IDs
INTEGER       dimids       ! list of returned dimension IDs
DOUBLEPRECISION  DVAL      ! single data value
DOUBLEPRECISION  dval      ! returned single data value
DOUBLEPRECISION  DVALS     ! array of data values
DOUBLEPRECISION  dvals     ! array of returned data values
INTEGER       FILLMODE     ! NF_NOFILL or NF_FILL, for setting fill mode
INTEGER*1     I1VAL        ! single data value
INTEGER*1     I1val        ! returned single data value
INTEGER*1     I1VALS       ! array of data values
INTEGER*1     i1vals       ! array of returned data values
INTEGER*2     I2VAL        ! single data value
INTEGER*2     i2val        ! returned single data value
INTEGER*2     I2VALS       ! array of data values
INTEGER*2     i2vals       ! array of returned data values
INTEGER       IMAP         ! index mapping vector
INTEGER       INDEX        ! variable array index vector
INTEGER       IVAL         ! single data value
INTEGER       ival         ! returned single data value
INTEGER       IVALS        ! array of data values
INTEGER       ivals        ! array of returned data values
INTEGER       LEN          ! dimension or attribute length
INTEGER       len          ! returned dimension or attribute length
INTEGER       MODE         ! open mode, one of NF_WRITE or NF_NOWRITE
CHARACTER(*)  NAME         ! dimension, variable, or attribute name
CHARACTER(*)  name         ! returned dim, var, or att name
INTEGER       natts        ! returned number of attributes
INTEGER       NCERR        ! error returned from NF_xxx function call
INTEGER       NCID         ! netCDF ID of an open netCDF dataset
INTEGER       ncid         ! returned netCDF ID
```

```
INTEGER       NCID_IN      ! netCDF ID of open source netCDF dataset
INTEGER       NCID_OUT     ! netCDF ID of open destination netCDF dataset
INTEGER       NDIMS        ! number of dimensions
INTEGER       ndims        ! returned number of dimensions
CHARACTER(*)  NEWNAME      ! new name for dim, var, or att
INTEGER       ngatts       ! returned number of global attributes
INTEGER       nvars        ! returned number of variables
INTEGER       old_mode     ! previous fill mode, NF_NOFILL or NF_FILL,
CHARACTER(*)  PATH         ! name of netCDF dataset
REAL          RVAL         ! single data value
REAL          rval         ! returned single data value
REAL          RVALS        ! array of data values
REAL          rvals        ! array of returned data values
INTEGER       START        ! variable array indices of first value
INTEGER       STRIDE       ! variable array dimensional strides
CHARACTER(*)  TEXT         ! input text value
CHARACTER(*)  text         ! returned text value
INTEGER       unlimdimid   ! returned ID of unlimited dimension
INTEGER       VARID        ! variable ID
INTEGER       varid        ! returned variable ID
INTEGER       VARID_IN     ! variable ID
INTEGER       VARID_OUT    ! variable ID
INTEGER       XTYPE        ! external type: NF_BYTE, NF_CHAR, ... ,
INTEGER       xtype        ! returned external type
```

Index

A

B

C

D

E

F

G

H

I

L

N

O

R

U

V