NetCDF User's Guide

An Interface for Data Access
Version 2.3, April 1993

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

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

Foreword

Unidata is a National Science Foundation-sponsored program empowering U.S. universities, through innovative applications of computers and networks, to make the best use of atmospheric and related data for enhancing education and research. The Unidata Program Center currently offers universities several (supported) software packages for analyzing and displaying such data, some developed by Unidata and some by other organizations, including NASA, the University of Wisconsin, Purdue University, and MacDonald Dettwiler. Underlying these is a Unidata- developed system for acquiring and managing data in real time, making practical the Unidata principle that each university should acquire and manage its own data holdings as local requirements dictate. (It is significant that the Unidata program has no data center – the management of data is a "distributed" function.)

The Network Common Data Form (netCDF) software described in this guide was originally intended to provide a common data access method for the various Unidata applications. These deal with a variety of data types that encompass single-point observations, time series, regularly-spaced grids, and satellite or radar images.

The netCDF software functions as an I/O library, callable from C or FORTRAN, which stores and retrieves scientific data structures in self- describing, machine-independent files. Each netCDF file can contain an unlimited number of multi-dimensional, named variables (with differing types that include integers, reals, characters, bytes, etc.), and each variable may be accompanied by ancillary data, such as units of measure or descriptive text. The interface includes a method for appending data to existing netCDF files in prescribed ways, functionality that is not unlike a (fixed length) record structure. However, the netCDF library also allows direct-access storage and retrieval of data by variable name and index and therefore is useful only for disk-resident (or memory- resident) files.

NetCDF access has been implemented in about half of Unidata's software, so far, and it is planned that such commonality will extend across all Unidata applications in order to:

- Facilitate the use of common data files by distinct applications.
- Permit data files to be transported between or shared by dissimilar computers transparently, i.e., without translation.
- Reduce the programming effort usually spent interpreting formats in a way that is equally effective for FORTRAN and C programmers.
- Reduce errors arising from misinterpreting data and ancillary data.
- Facilitate using output from one application as input to another.

- Establish an interface standard which simplifies the inclusion of new software into the Unidata system.

A measure of success has been achieved. The netCDF is now in use on computing platforms that range from CRAYs to Personal Computers and include many (UNIX- and VMS-based) workstations. It can be used to create a complex data set on one computer (say in FORTRAN) and retrieve that same (self-describing) data set on another computer (say in C) without intermediate translations – netCDF files can be transferred across a network, or they can be accessed remotely using a suitable network file system.

Because we believe that the use of netCDF access in non-Unidata software will benefit Unidata's primary constituency – such use may result in more options for analyzing and displaying Unidata information – the netCDF library is distributed without licensing or other significant restrictions, and current versions can be obtained via anonymous FTP. Apparently the software has been well received by a wide range of institutions beyond the atmospheric science community, and a substantial number of public domain and commercial data analysis systems can now accept netCDF files as input.

Several organizations have adopted the netCDF as a data access standard, and there is an effort underway at the National Center for Supercomputer Applications (NCSA, which is associated with the University of Illinois at Urbana-Champaign) to support the netCDF programming interfaces as a means to store and retrieve data in "HDF files," i.e., in the format used by the popular NCSA tools. We have encouraged and cooperated with these efforts.

Questions occasionally arise about the level of support provided for the netCDF software. Unidata's formal position, stated in the copyright notice which accompanies the netCDF library, is that the software is provided "as is" ... In practice, the software is updated from time to time, and Unidata intends to continue making improvements for the foreseeable future. Because Unidata's mission is to serve atmospheric scientists at U.S. universities, problems reported by that community necessarily receive the greatest attention.

We hope the reader will find the software useful and will give us feedback on its application as well as suggestions for its improvement.

David Fulker
Unidata Program Center Director University Corporation for Atmospheric Research

Summary

The purpose of the Network Common Data Form (netCDF) interface is to allow you to create, access, and share scientific data in a form that is self-describing and network-transparent. "Self-describing" means that a file includes information defining the data it contains. "Network-transparent" means that a file is represented in a form that can be accessed by computers with different ways of storing integers, characters, and floating-point numbers. Using the netCDF interface for creating new scientific data sets can improve the accessibility of the data. Using the netCDF interface in new software for scientific data access, management, analysis, and display can improve the reusability of the software for other data sets and by other users.

The netCDF software provides common C and FORTRAN interfaces for applications and data. The C interface library is available for many common computing platforms, including various versions of UNIX, VMS, MSDOS and OS/2 environments. The FORTRAN interface, implemented in C, is available for many of the platforms on which the C interface is available. The form in which the FORTRAN interface is implemented makes it easy to generate and test a new FORTRAN interface for a new platform by specifying the conventions for calling C from FORTRAN. An experimental C++ prototype interface is currently under evaluation.

The netCDF software is being made freely available to encourage the sharing of both scientific data and the software that makes the data useful.

1 Introduction

1.1 The NetCDF Interface

The Network Common Data Form, or netCDF, is an interface to a library of data access programs for storing and retrieving scientific data. NetCDF is an abstraction that supports a view of data as a collection of self-describing, network-transparent objects that can be accessed through a simple interface. Collections of named multidimensional variables can be randomly accessed, without knowing details of how the data are stored. Auxiliary information about the data, such as what units are used, can be stored with the data. Generic utilities and application programs can be written that access arbitrary netCDF files and transform, combine, analyze, or display specified fields of the data. The development of such applications may lead to improved accessibility of data and improved reusability of software for scientific data management, analysis, and display.

The netCDF software implements an *abstract data type*, which means that all operations to access and manipulate data in a netCDF file must use only the set of functions provided by the interface. The actual representation of the data is hidden from applications that use the interface, so that how the data are stored could be changed without affecting such programs. The physical representation of netCDF data is designed to be independent of the computer on which the data were written. Future changes to the netCDF interface will be compatible with the interface described here, so that neither existing netCDF files nor programs accessing them will require modification.

The netCDF interface is supported for both C and FORTRAN, and for UNIX, VMS, OS/2 and MSDOS operating systems. The netCDF software that implements this interface is freely available via FTP and free of licensing restrictions to encourage its wide use.

1.2 NetCDF is Not a Database Management System

Why not use an existing database management system (DBMS) for storing scientific data? We looked at available database software, both commercial and research-oriented, and concluded that existing packages are currently inadequate for the kinds of scientific data access supported by the netCDF interface.

First, most existing DBMSs have poor support for multidimensional objects as the basic unit of data access. An alternative data model of comparable power and elegance to the relational model for databases is needed for scientific data. If such a model existed, programs that shared that model could be used in flexible combinations to support effective systems for building applications and visualizing data. Representing multidimensional arrays as relations makes some useful kinds of data access awkward and provides little support for the abstractions of multidimensional data and coordinate systems.

Related to this is a second problem with general-purpose database systems: their poor performance on large scientific data sets. Collections of satellite images, scientific model outputs, climate observations covering decades, high-resolution atmospheric profile data, and other large data sets are beyond the capabilities of most DBMSs to organize and index for efficient retrieval.

Finally, general-purpose database systems provide, at significant cost in terms of both resources and access performance, many facilities that are not needed in the analysis, management, and display of scientific data. For example, elaborate update facilities, concurrency control, audit trails, report writers, and mechanisms designed for transaction-processing are unnecessary for most scientific applications.

1.3 What about Performance?

To achieve network-transparency, the netCDF is implemented on top of a layer of software for external data representation (XDR). XDR, developed by Sun Microsystems, Inc., is a nonproprietary standard for describing and encoding data. It supports encoding arbitrary C data structures into machine-independent sequences of bits. The encoding used for floating-point numbers is the Institute for Electrical and Electronics Engineers (IEEE) standard for normalized floating-point numbers. XDR has been implemented on a wide variety of computers, including Suns, VAXs, Apple Macintoshes, IBM RS 6000s, IBM PS/2s, IBM mainframes, and CRAYs. It assumes only that eight-bit bytes can be encoded and decoded in a consistent way.

Translating data into and out of XDR form adds overhead to data transfers, but for many applications the extra CPU cycles used to convert data to and from a machine-independent representation are not significant. The amount of XDR overhead depends on many factors, including the data type, the type of computer, the granularity of data access, and how well the implementation has been tuned to the computer on which it is run. For a large set of applications, the overhead of the XDR layer is a reasonable price to pay for portable, network-transparent data access.

Often when an abstraction layer is added to hide the details of an underlying implementation, some computations that can be expressed simply in terms of the abstraction may be computationally expensive. Furthermore, it may not be obvious which of several ways of expressing a computation through the abstract interface will make efficient use of computing resources, without understanding something about the implementation. It is certainly possible to use the netCDF interface to access data in inefficient ways: for example, by requesting a slice of variable data that requires a single value out of each record. See ⟨undefined⟩ [NetCDF File Structure and Performance], page ⟨undefined⟩, for a discussion of performance characteristics of the implementation and how to apply knowledge of the underlying implementation to use the interface effectively when performance is an important concern.

1.4 Is NetCDF a Good Archive Format?

NetCDF can be used as an archive format for storing data, but it may take more space than a special-purpose archive format that exploits knowledge of particular characteristics of a set of data. Compression of data is possible with netCDF (e.g., using arrays of eight-bit bytes to encode low-resolution floating-point numbers instead of arrays of 32-bit numbers), but netCDF was not designed to achieve optimal compression of scientific data.

The advantages of a special-purpose archive format for small archives should be compared to the benefits of machine-independence and the ability to store ancillary data (data about the data) that the netCDF interface provides. For large archives, only two programs need to be provided for each archive format, one to translate archived data into netCDF form and the other to translate back to the archive format. Tools provided for manipulating netCDF data will then be available without sacrificing the advantages of the archive format and without requiring the wholesale conversion of large existing archives.

1.5 Background and Evolution of the NetCDF Interface

The development of the netCDF interface began with a modest goal related to Unidata's needs: to provide a common interface between Unidata applications and ingested real-time meteorological data. Since Unidata software was intended to run on multiple hardware platforms with access from both C and FORTRAN, achieving Unidata's goals had the potential for providing a package that was useful in a broader context. By making the package widely available and collaborating with other organizations with similar needs, we hoped to improve the current situation in which scientific software is only rarely reused by others in the same discipline and almost never reused between disciplines (Fulker, 1988).

Important concepts employed in the netCDF software originated in a paper (Treinish and Gough, 1987) that described data-access software developed at the NASA Goddard National Space Science Data Center (NSSDC). The interface provided by this software was called the Common Data

Format (CDF). The NASA CDF was originally developed as a platform-specific FORTRAN library to support an abstraction for storing multidimensional scientific data.

The NASA CDF package had been used for many different kinds of data in an extensive collection of applications. It had the virtues of simplicity (only 13 subroutines), independence from storage format, generality, ability to support logical user views of data, and support for generic applications.

Unidata held a workshop on CDF in Boulder in August 1987. We proposed exploring the possibility of collaborating with NASA to extend the CDF FORTRAN interface, to define a C interface, and to permit the access of data aggregates with a single call, while maintaining compatibility with the existing NASA interface.

Independently, Dave Raymond at the New Mexico Institute of Mining and Technology had developed a package of C software for UNIX that supported self-describing scientific data along with a "pipes and filters" approach to processing, analyzing, and displaying scientific data. This package also used the "Common Data Format" name, later changed to C-Based Analysis and Display System (CANDIS). Unidata learned of Raymond's work (Raymond, 1988), and incorporated some of his ideas, such as the use of named dimensions and variables with differing shapes in a single data object, into the Unidata netCDF interface.

In early 1988, Glenn Davis of Unidata developed a prototype netCDF package in C that was layered on a nonproprietary external data representation standard (XDR) developed by Sun Microsystems. This prototype proved that a single-file, network-transparent implementation of the CDF interface could be achieved at acceptable cost and that the resulting programs could be implemented on both UNIX and VMS systems. However, it also demonstrated that providing a small, portable, and NASA CDF-compatible FORTRAN interface with the desired generality was not practical. (NASA's CDF and Unidata's netCDF have since evolved separately, but recent CDF versions share many characteristics with netCDF.)

In early 1988, Joe Fahle of SeaSpace, Inc. (a commercial software development firm in San Diego, California), a participant in the 1987 Unidata CDF workshop, independently developed a CDF package in C that extended the NASA CDF interface in several important ways (Fahle, 1989). Like Raymond's package, the SeaSpace CDF software permitted variables with unrelated shapes to be included in the same data object and permitted a general "hyperslab" form of access to multidimensional arrays. Fahle's implementation was used at SeaSpace as the intermediate form of storage for a variety of steps in their image-processing system.

After studying Fahle's interface, we concluded that it solved many of the problems we had identified in trying to stretch the NASA interface to our purposes. In August 1988, we convened a small workshop to agree on a Unidata netCDF interface, and to resolve remaining open issues. Attending were Joe Fahle of SeaSpace, Michael Gough of Apple (an author of the NASA CDF software), Angel Li of the University of Miami (who had implemented our prototype netCDF software on VMS and was a potential user), and Unidata systems development staff. Consensus was reached at the workshop after some further simplifications were discovered. A document incorporating the results of the workshop into a proposed Unidata netCDF interface specification was distributed widely for comments before Glenn Davis implemented the software it described. Comparison with other data-access interfaces and recent experience in using netCDF are discussed in (Rew and Davis, 1990a) and (Rew and Davis, 1990b).

In October 1991, we announced version 2.0 of the netCDF software distribution. Slight modifications to the C interface (declaring dimension sizes to be `long` rather than `int`) improved the usability of netCDF on inexpensive platforms such as MSDOS computers, without requiring recompilation on other platforms. This change to the interface required no changes to the associated file format.

1.6 What's New in this Release?

This Guide documents the March 1993 release of netCDF 2.3, which preserves the same file format but adds a few new functions to the C and Fortran interfaces.

The suggested extension for netCDF files has been changed from `.cdf` to `.nc`, in order to avoid a clash with the NASA CDF file extension. Although the old extension is still supported in a backward compatible way, new netCDF files should use the new filename extension, where practical.

Subsampling along specified dimensions (using 'strides') is one of the commonly-asked-for features supported by the new `ncvarputg()` and `ncvargetg()` interfaces for generalized hyperslab access. In addition, these interfaces permit accessing data that is not contiguous in memory. In a generalized hyperslab, an index mapping vector is used to define the mapping between points in the generalized hyperslab and the memory locations of the corresponding values. (See ⟨undefined⟩ [ncvarputg NCVPTG and NCVPGC], page ⟨undefined⟩.)

There are also some new interfaces that can be used to write, read, and inquire about records, where a record may contain multiple variables of different types and shapes. Where before you had to access a record's worth of data using multiple calls, now you may instead use a single call. (See ⟨undefined⟩ [ncrecput], page ⟨undefined⟩.)

New optimizations for the library have resulted in significant speedups for accessing cross-sections involving non-contiguous data. Many bugs have been fixed, portability has been improved, and the installation has been greatly simplified for most systems.

The ncdump utility now supports several new command-line options including the ability to specify for which variables data values will be output, to provide brief annotations in the form of CDL comments to identify data values for large multidimensional variables, or to provide full annotations in the form of trailing CDL comments for every data value.

1.7 Future Plans for NetCDF

We continue to use netCDF for Unidata system software and applications. We have begun to use netCDF interfaces for earth-referenced image-analysis software, as well as for the output of decoders for a wide variety of meteorological and oceanographic data.

We have recently released a preliminary set of netCDF operators that we hope will eventually be completed to provide an algebra of useful operations on generic scientific data stored in netCDF files. The complete set of operators includes selectors to extract subsets of variables or reduce the dimensionality of a netCDF file; combiners to merge netCDF files, combine variables, or increase dimensionality; graphics generators to read netCDF files and produce graphical output; mathematical operators; and specialized data converters to convert units, convert to or from standard archive forms, or to convert to a canonical form for comparison. By composing these fundamental operators, users would have a wide variety of capabilities.

We have submitted a proposal for resources to extend the netCDF library in an upward-compatible way to support netCDF servers on a network. With netCDF servers, clients could access cross-sections of data efficiently, as if they were stored in a local file. A netCDF server would also have the ability to support virtual netCDF objects that provide different views of large or remote data sets. Servers might also be capable of providing a transparent netCDF interface to non-netCDF archives.

Other desirable extensions that we would like to see added to netCDF include an interface for record access by key or coordinate value, transparent data packing by use of three new reserved attributes (`_Nbits`, `_Scale`, and `_Offset`), support for pointers to data cross-sections in other files (what Joe Fahle calls "assemblies"), better string support, and a well-tested C++ interface. Some of these extensions may be included in future releases.

References

1. Fahle, J., *TeraScan Applications Programming Interface*, SeaSpace, San Diego, California, 1989.
2. Fulker, D. W., "The netCDF: Self-Describing, Portable Files—a Basis for 'Plug-Compatible' Software Modules Connectable by Networks," ICSU Workshop on Geophysical Informatics, Moscow, USSR, August 1988.
3. Fulker, D. W., "Unidata Strawman for Storing Earth-Referencing Data," *Seventh International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, New Orleans, La., American Meteorology Society, January 1991.
4. Gough, M. L., *NSSDC CDF Implementer's Guide (DEC VAX/VMS) Version 1.1*, National Space Science Data Center, 88-17, NASA/Goddard Space Flight Center, 1988.
5. Jenter, H. L. and R. P. Signell, "NetCDF: A Freely-Available Software-Solution to Data-Access Problems for Numerical Modelers," Proceedings of the American Society of Civil Engineers Conference on Estuarine and Coastal Modeling, Tampa, Florida, 1992.
6. Raymond, D. J., "A C Language-Based Modular System for Analyzing and Displaying Gridded Numerical Data," Journal of Atmospheric and Oceanic Technology, **5**, 501-511, 1988.
7. Rew, R. K. and G. P. Davis, "The Unidata netCDF: Software for Scientific Data Access," *Sixth International Conference on Interactive Information and Processing Systems for Meteorology, Oceanography, and Hydrology*, Anaheim, California, American Meteorology Society, February 1990.
8. Rew, R. K. and G. P. Davis, "NetCDF: An Interface for Scientific Data Access," *Computer Graphics and Applications*, IEEE, pp. 76-82, July 1990.
9. Treinish, L. A. and M. L. Gough, "A Software Package for the Data Independent Management of Multi-Dimensional Data," *EOS Transactions*, American Geophysical Union, **68**, 633-635, 1987.
10. Sun Microsystems, "External Data Representation Standard: Protocol Specification," RFC 1014; Information Sciences Institute, May 1988.

2 Components of a NetCDF File

A netCDF file has *dimensions*, *variables*, and *attributes*. These components can be used together to capture the meaning of data and relations among data fields in a scientific data set.

We will use a small netCDF example to illustrate the concepts of netCDF dimensions, variables, and attributes. The notation used to describe this simple netCDF object is called CDL (network Common Data form Language). It provides an easily comprehended text version of the structure and contents of a binary netCDF file:

```
netcdf example_1 {  // example of CDL notation for a netCDF file

dimensions:          // dimension names and sizes are declared first
        lat = 5, lon = 10, level = 4, time = unlimited;

variables:           // variable types, names, shapes, attributes
        float   temp(time,level,lat,lon);
                    temp:long_name     = "temperature";
                    temp:units         = "celsius";
        float   rh(time,lat,lon);
                    rh:long_name = "relative humidity";
                    rh:valid_range = 0.0, 1.0;      // min and max
        int     lat(lat), lon(lon), level(level);
                    lat:units       = "degrees_north";
                    lon:units       = "degrees_east";
                    level:units     = "millibars";
        short   time(time);
                    time:units      = "hours since 1990-11-25 12:00 UTC";
        // global attributes
                    :source = "National Weather Service";

data:                // optional data assignments
        level   = 1000, 850, 700, 500;
        lat     = 20, 30, 40, 50, 60;
        lon     = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
        time    = 12;
        rh      =.5,.2,.4,.2,.3,.2,.4,.5,.6,.7,
                 .1,.3,.1,.1,.1,.1,.5,.7,.8,.8,
                 .1,.2,.2,.2,.2,.5,.7,.8,.9,.9,
                 .1,.2,.3,.3,.3,.3,.7,.8,.9,.9,
                  0,.1,.2,.4,.4,.4,.4,.7,.9,.9;
}
```

The CDL notation for a netCDF file can be generated automatically by using `ncdump`, a utility program described later (see ⟨undefined⟩ [ncdump], page ⟨undefined⟩). Another netCDF utility, `ncgen`, generates a netCDF file (or optionally C or FORTRAN source code containing calls needed to produce a netCDF file) from CDL input (see ⟨undefined⟩ [ncgen], page ⟨undefined⟩). It is not necessary to learn much about CDL notation to use the netCDF library; we use it in this document as a concise way of presenting netCDF examples.

The CDL notation will be explained more fully as we describe the components of a netCDF file. For now, note that CDL statements are terminated by a semicolon. Spaces, tabs, and newlines can be used freely for readability. Comments in CDL follow the characters '`//`' on any line. A CDL description of a netCDF file takes the form

```
netCDF name {
  dimensions: ...
  variables: ...
  data: ...
}
```

where the *name* is used only as a default in constructing the name of the file generated by the `ncgen` utility. The CDL description consists of three optional parts, introduced by the keywords `dimensions`, `variables`, and `data`. NetCDF dimension declarations appear after the `dimensions` keyword, netCDF variables and attributes are defined after the `variables` keyword, and variable data assignments appear after the `data` keyword.

2.1 Dimensions

A netCDF dimension is a named integer used to specify the shape of one or more of the multi-dimensional variables contained in a netCDF file. A dimension may be used to represent a real physical dimension, for example, time, latitude, longitude, or height. A dimension might also be used to index more abstract quantities, for example, color-table entry number, instrument number, station-time pair, or model-run ID.

Every netCDF dimension has both a *name* and a *size*. A dimension name is an arbitrary sequence of alphanumeric characters (as well as the underscore character, '`_`', and the hyphen character, '`-`') beginning with a letter. Case is distinguished in netCDF names. A dimension size is an arbitrary positive integer, except that one dimension in a netCDF file can have the size `UNLIMITED`. Such a dimension is called the *unlimited dimension* or the *record dimension*. A variable with an unlimited dimension can grow to any length along that dimension. The unlimited dimension is like a record number in conventional record-oriented files.

CDL dimension declarations may appear on one or more lines following the CDL keyword `dimensions`. Multiple dimension declarations on the same line may be separated by commas. Each declaration is of the form *name* = *size*.

There are four dimensions in the example: `lat`, `lon`, `level`, and `time`. The first three are assigned fixed sizes; `time` is assigned the size `UNLIMITED`, which means it is the *unlimited* dimension. A netCDF file can have at most one unlimited dimension, but need not have any.

There are several uses for netCDF dimensions:

- Specifying the shapes and sizes of variables.
- Identifying and relating variables that are defined on a common grid.
- Providing a way to define coordinate systems.

We discuss each of these uses below.

2.1.1 Using Dimensions to Specify Variable Shapes

The basic unit of named data in a netCDF file is a *variable*. In general, a variable is a multidimensional object that has, among other characteristics, a *shape*, which is defined by the number, order, and sizes of its dimensions. When a netCDF variable is defined, the number and order of the dimensions that define its shape are specified. Hence you must first create the necessary dimensions before creating a netCDF variable that uses them.

It is possible to use the same dimension more than once in specifying a variable shape, for example `var(dim, dim)`, but it does not make much sense to do this; it is contrary to the intuitive meaning of a physical dimension. A variable that has two dimensions that happen to be the same size is more accurately modeled by using two dimensions with different names but the same size.

2.1.2 Using Dimensions to Relate Variables

Two dimensions may have the same size, perhaps by coincidence, without being related in any other way. Dimension names provide a way to distinguish dimensions regardless of size.

Variables are related by the dimensions they share. For example, if two variables are defined with the same dimensions, they might represent observations or model output for the same set of points. In the example netCDF file, variables `temp` and `rh` share the `time`, `lat`, and `lon` dimensions. The variable `rh` does not have `level` as a dimension, perhaps because it is only defined for a single level.

2.1.3 Using Dimensions to Define Coordinate Systems

Besides serving as sizes for integer indexes to multidimensional variables, dimensions may be used to define coordinate systems for variable data. To do this, create a variable with the same name as a dimension and specify coordinate values for that variable. A variable should only be given the same name as a dimension in a netCDF file when it is to be used as a *coordinate* variable. Such variables are indexed by the dimension for which they provide coordinate values, for example, `lat(lat)`.

It is not necessary to provide a coordinate variable for each dimension; if no such variable is defined, the coordinate values of the dimension are assumed to be 0, 1, 2, ... (for C programs) or 1, 2, 3, ... (for FORTRAN programs). Although the C and FORTRAN interfaces support different conventions for index numbering, there is no difference between the actual netCDF files written by C and FORTRAN programs. Programs written in either language can be used to access data written by programs using the other interface.

In the CDL example, each dimension has an associated coordinate variable with the same name as the dimension. The four values of the `level` index, 0, 1, 2, 3, (1, 2, 3, 4 in FORTRAN) are related in coordinate-like fashion to the four values (1000, 850, 700, 500) of the `level` variable. Note that there is no requirement that coordinates be equally spaced or increasing. It would not make much sense for two coordinate values to be the same, but the meaning of coordinate variables is enforced only by conventions of application packages and utilities, not by the netCDF interface. The application packages and utilities that make use netCDF coordinate variables assume they are one-dimensional and monotonic (all values either increasing or decreasing).

2.2 Variables

A *variable* represents a multidimensional array of values of the same type. A variable has a name, a data type, and a shape described by its list of dimensions, all of which are specified when the variable is created. Each variable may also have data values and associated attributes, which may be added or changed after the variable is created. Variables are used to store the bulk of the data in a netCDF file, and are the primary component used by utilities to identify sub-parts of a netCDF file.

Like a dimension name, a variable name is an arbitrary sequence of alphanumeric characters (also including '`_`' and '`-`') beginning with a letter. Case is distinguished in variable names. Long names help to make a netCDF file self-documenting, but ancillary information about a variable is better stored in variable *attributes* (discussed below) than encoded as part of the name.

A variable data type is one of a small set of netCDF *types* that have the names `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE` in the C interface and the corresponding names `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE` in the FORTRAN interface. In the CDL notation, these types are given the simpler names `byte`, `char`, `short`, `long`, `float`, and `double`. `int` may be used as a synonym for `long` and `real` may be used as a synonym for `float` in the CDL notation. We will postpone a discussion of the exact meaning of each of the types until the discussion of *data*, below. For now, it suffices to know that the choice of the type used to represent variable data depends on the range of values it can have, the precision to which values are known, and the number of bits required to represent the variable in a netCDF file on disk.

The shape of a variable is specified by its list of dimensions. If a variable has an unlimited dimension, that dimension must appear first in the list of dimensions in CDL. It is possible to define variables with no dimensions, also called *scalar* variables. There are no scalar variables in the example netCDF file.

CDL variable declarations appear after the `variables` keyword in a CDL unit. They have the form

type variable_name (*dim_name_1*, *dim_name_2*, ...) ;

for variables with dimensions, or

type variable_name ;

for scalar variables.

In the CDL example there are six variables. As discussed above, four of these are coordinate variables for dimensions. The remaining variables, `temp` and `rh`, contain what is usually thought of as the data. Each of these variables has the unlimited dimension `time` as its first dimension, so they are called *record variables*. A variable that is not a record variable has a fixed size (number of data values) given by the product of its dimensions. A record variable has a current size, given by the product of the maximum record written so far and the other dimensions of the variable. Only record variables may grow after they are defined.

2.3 Attributes

A netCDF *attribute* is meant to contain information about a netCDF variable or about an entire netCDF file. This information is *ancillary data*, or data about data, analogous to the information stored in data dictionaries and schema in conventional database systems. An attribute has an associated variable, a name, a data type, a length, and a value. Individual attributes are identified by specifying a variable and an attribute name.

Each attribute is associated with a single variable when it is created. Attributes for different variables may differ in data type, length, and values even though they share the same name.

A *global* attribute is one that applies to the whole netCDF rather than any particular variable. Global attributes are defined and accessed similarly to variable attributes; the details for defining global attributes in the CDL notation and in the netCDF procedural interface are presented later.

Attribute names follow the same rules as dimension and variable names. Providing meaningful names for attributes is important, but using agreed on conventional names is also required if generic applications and utility programs will be used on a netCDF file. For example, every variable for which units make sense should have a `units` attribute defined, so the units can be printed in labels. Furthermore, if the netCDF file is ever to be used as input to generic units-aware software, the values of the `units` attributes should be expressed in a conventional form as a character string that can be interpreted by that software.

The type of an attribute is specified when it is created. The types permitted for attributes are exactly the same as the netCDF data types used in creating variables. Attributes with the same name for different variables should sometimes be of different types. For example, the attribute `valid_max` specifying the maximum valid data value for a variable of type `long` should be of type `long`, whereas the attribute `valid_max` for a variable of type `double` should instead be of type `double`.

In addition to specifying the associated variable, attribute name, and type, the length and value of an attribute must also be specified when it is created. The information in an attribute is represented by either a single value (length 1) or a vector of values of the same type. Since "character string" is not a basic netCDF data type, string-valued attributes have a vector of characters as their value, with a length equal to the length of the character string.

Attributes are more dynamic than variables or dimensions; they can have their type, length, and values changed after they are created. For example, an attribute `max_value` might store the maximum value seen so far for a record variable, and might be updated every time a new record is written.

The CDL notation for defining an attribute is

variable_name:attribute_name = *list_of_values* ;

for a variable attribute, or

:attribute_name = *list_of_values* ;

for a global attribute. The type and length of each attribute are not explicitly declared in CDL; they are derived from the values assigned to the attribute. All values of an attribute must be of the same type. The notation used for constant values of the various netCDF types is discussed later (see ⟨undefined⟩ [CDL Constants], page ⟨undefined⟩).

In the example netCDF (see ⟨undefined⟩ [CDL example], page ⟨undefined⟩), `units` is an attribute for the variable `lat` that has a 13 character array value '`degrees_north`'. And `valid_range` is an attribute for the variable `rh` that has length 2 and values '`0.0`' and '`1.0`'.

One global attribute—`source`—is defined for the example netCDF file. This is a character array intended for documenting the data. Real netCDF files typically have more global attributes to document the origin, history, accuracy, and other characteristics of the data.

2.3.1 Attribute Conventions

Generic applications that take netCDF files as input may by convention expect certain variable and global attributes. A few other attributes are handled in special ways by the netCDF library (these reserved attributes will have names that begin with a leading underscore character '`_`'). If you want to be able to use generic applications with your files, you should use the following conventional names for these commonly used attributes:

`units` A character array that specifies the units used for the variable's data. A standard for conventional ways to name units in each specific discipline should be used, if available. Unidata has developed a freely-available library of routines to convert between character string and binary forms of unit specifications and to perform various useful operations on the binary forms. This library is used in some netCDF applications. Using the recommended units syntax permits data represented in conformable units to be automatically converted to common units for algebraic operations. See ⟨undefined⟩ [Units], page ⟨undefined⟩, for more information.

`long_name`
: A long descriptive name. This could be used for labelling plots, for example. If a variable has no `long_name` attribute assigned, the variable name will be used as a default.

`valid_range`
: An array of two numbers specifying the minimum and maximum valid values for this variable. The type of each `valid_range` attribute should match the type of its variable.

`valid_min`
`valid_max`
: One or both of these may be used instead of `valid_range`; this handles the case where it only makes sense to bound the data below or above.

`scale_factor`
: If present for a variable, the data are to be multiplied by this factor after the data are read by the application that accesses the data.

`add_offset`
: If present for a variable, this number is to be added to the data after it is read by the application that accesses the data. If both `scale_factor` and `add_offset` attributes are present, the data are first scaled before the offset is added. The attributes `scale_factor` and `add_offset` can be used together to provide simple data compression to store low-resolution floating-point data as small integers in a netCDF file. When scaled data are written, the application should first subtract the offset and then divide by the scale factor.

When `scale_factor` and `add_offset` are used for packing, the associated variable (containing the packed data) is typically of type byte or short, whereas the unpacked values are intended to be of type float or double. The attributes `scale_factor` and `add_offset` should both be of the type intended for the unpacked data, e.g. float or double.

`_FillValue`

If a scalar attribute with this name is defined for a variable and is of the same type as the variable, it will be subsequently used as the *fill value* for that variable. The purpose of this attribute is to save the applications programmer the work of prefilling the data and also to eliminate the duplicate writes that result from netCDF filling in missing data with its default fill value, only to be immediately overwritten by the programmer's preferred value. This value is considered to be a special value that indicates missing data, and is returned when reading values that were not written. The missing value should be outside the range specified by `valid_range` for a variable. It is not necessary to define your own `_FillValue` attribute for a variable if the default *fill value* for the type of the variable is adequate. Note that if you change the value of this attribute, the changed value only applies to subsequent writes; previously written data are not changed. See ⟨undefined⟩ [Missing Values], page ⟨undefined⟩, for more information.

`missing_value`

`missing_value` is a conventional name for a missing value that will not be treated in any special way be the library, as the `_FillValue` attribute is. It is also useful when it is necessary to distinguish between two kinds of missing values. For example, `_FillValue` might be useful to indicate data that was expected but did not appear, whereas `missing_value` might be used to indicate grid regions that are not intended to contain data. See ⟨undefined⟩ [Missing Values], page ⟨undefined⟩, for more information.

`signedness`

Used to indicate a nondefault interpretation of the signedness of integer values. By default, applications that deal with values should treat netCDF byte data as unsigned and netCDF short and long integer data as signed. If you declare a netCDF variable for storing bytes, and you intend that the values represent signed quantities, you should declare the variable attribute `signedness` with value `"signed"`. Similarly, if you define a variable for an array of short or long integers and you intend that the values be interpreted as unsigned, it would be appropriate to define the variable attribute `signedness = "unsigned"`. This attribute is ignored by the netCDF library, but applications may use it. Since there are no standard FORTRAN types corresponding to unsigned integers, FORTRAN programs that compute with or use the ordering of data values may need to handle this attribute.

`C_format` A character array for the format that should be used to print values for this variable by C applications. For example, if you know a variable is only accurate to three significant digits, it would be appropriate to define the `C_format` attribute as `"%.3g"`. The `ncdump` utility program uses this attribute for variables for which it is defined.

`FORTRAN_format`
: A character array for the format that should be used to print values for this variable by FORTRAN applications.

`title`
: A global attribute that is a character array providing a succinct description of what is in the data set.

`history`
: A global attribute that is a character array with a line for each invocation of a program and arguments that were used to derive the file. Well-behaved generic netCDF filters (programs that take netCDF files as input and produce netCDF files as output) will automatically append their name and the parameters with which they were invoked to the global history attribute of an input netCDF file.

Attributes may be added to a netCDF file long after it is first defined, so you don't have to anticipate all potentially useful attributes. However adding new attributes to an existing file can incur the same expense as copying the file. See ⟨undefined⟩ [NetCDF File Structure and Performance], page ⟨undefined⟩, for a more extensive discussion.

2.3.2 Differences between Attributes and Variables

In contrast to variables, which are intended for bulk data, attributes are intended for ancillary data, or information about the data. Typically the data in variables of an open netCDF will reside on disk, because the data are too large to fit in memory all at once. In contrast, the total amount of ancillary data associated with a netCDF object and stored in its attributes is typically small enough to be memory-resident.

Another difference between attributes and variables is that variables may be multidimensional. Attributes are all either scalars (single-valued) or vectors (a single, fixed dimension).

Variables are created with a name, type, and shape before they are assigned data values, so a variable may exist with no values. The value of an attribute must be specified when it is created, so no attribute ever exists without a value.

A variable may have attributes, but an attribute cannot have attributes. Attributes assigned to variables may have the same units as the variable (for example, `valid_range`) or have no units (for example, `scale_factor`). If you want to store data in a netCDF file that requires units different from those of the associated variable, it is better to use a variable than an attribute. More generally, if data require ancillary data to describe them, are multidimensional, require any of the defined netCDF dimensions to index their values, or require a significant amount of storage, the data should be represented using variables rather than attributes.

3 Data

This chapter discusses the six primitive netCDF data types, the kinds of data access supported by the netCDF interface, and how data structures other than multidimensional arrays may be implemented in a netCDF file.

3.1 NetCDF Data Types

The current set of primitive types supported by the netCDF interface are:

`byte` Eight-bit data; especially good for saving space when only a few values are possible or resolution is low.

`character`
Currently synonymous with `byte`; intended for representing text strings as arrays of ASCII characters.

`short` 16-bit integers.

`long` 32-bit integers.

`float` 32-bit IEEE floating-point.

`double` 64-bit IEEE floating-point.

Except for the added `byte` and the lack of unsigned types, netCDF supports the same primitive data types as C. The names for the primitive data types are reserved words in CDL, so the names of variables, dimensions, and attributes must not be type names. Whether byte, short, or long data is interpreted as signed or unsigned is not part of the netCDF interface; since no netCDF operations depend on the sign or order of variable data, you are free to interpret a byte, for example, as holding values between 0 and 255 or between -128 and 127. For convenience, short and long constants are interpreted as signed in the CDL notation. See ⟨undefined⟩ [Attribute Conventions], page ⟨undefined⟩ for more information on representing signedness of values.

These types were chosen because they are familiar to C and FORTRAN programmers, they have well-defined external representations independent of any particular computers (using XDR), and they are sufficient for providing a reasonably wide range of trade-offs between data precision and number of bits required for each datum.

Additional primitive types may be added in the future, but only in a way that is compatible with existing programs and files. For example, `hyperlong` for 64-bit integers will eventually be needed, along with a new type for multibyte characters, but these can both be added without affecting existing netCDF files or applications, and with only minor changes required for generic applications that will support them.

3.2 Data Access

The netCDF interface supports several kinds of data access to data in an open netCDF file:

- direct (random) access to single data values,
- direct access to an arbitrary generalized cross-section of data for a single variable,
- record-oriented access to data for a single variable, and
- record-oriented access to a subset of the variables in a record.

To directly access a single data value, you specify a netCDF file, a variable, and a multidimensional index for the variable. Files are not specified by name every time you want to access data, but instead by a small integer obtained when the file is first created or opened. Similarly, variables are not specified by name for every data access either, but by variable IDs, small integers used to identify variables in a netCDF file.

Data in a netCDF file can be accessed as single values or as *hyperslabs*. A hyperslab is a generalized piece of a multidimensional variable that is specified by giving the indices of a corner point and a list of edge lengths along each of the dimensions of the variable. The corner point specified must be the one with the smallest indices, that is the one closest to the origin of the variable index space. The block of data values returned (or written) has the last dimension of the variable varying fastest, and the first dimension varying most slowly in the C interface. For FORTRAN, the order is reversed, with the first dimension of the variable varying fastest and the last dimension varying most slowly. These ordering conventions correspond to the customary order in which multidimensional variables are stored in C and FORTRAN.

As an example of hyperslab access, assume that in the example netCDF file (see ⟨undefined⟩ [CDL example], page ⟨undefined⟩), you wish to read all the data for the `temp` variable at the second (850 millibar) level, and assume that there are currently three records (`time` values) in the netCDF file. Recall that the dimensions are defined as

```
lat = 5, lon = 10, level = 4, time = unlimited;
```

and the variable `temp` is declared as

```
float   temp(time, level, lat, lon);
```

in the CDL notation.

A corresponding C variable that holds data for all four levels and all three times might be declared as

```
#define LATS  5
#define LONS 10
#define LEVELS 4
#define TIMES 3                  /* currently */
    ...
float   temp[TIMES*LEVELS*LATS*LONS];
```

to keep the data in a one-dimensional array, or

```
    ...
float   temp[TIMES][LEVELS][LATS][LONS];
```

using a multidimensional array declaration. To read in a hyperslab of data for only one level at a time, for example, you could leave out the LEVEL dimension or set it to 1.

In FORTRAN, the dimensions are reversed from the CDL declaration with the first dimension varying fastest and the record dimension as the last dimension of a record variable. Thus a FORTRAN declaration for the coresponding variable that holds all times and levels is

```
PARAMETER (LATS=5, LONS=10, LEVELS=4, TIMES=3)
   ...
REAL TEMP(LONS, LATS, LEVELS, TIMES)
```

To specify the hyperslab of data that represents the second level, all times, all latitudes, and all longitudes, we need to provide a corner and some edge lengths. The corner should be (0, 1, 0, 0) in C—or (1, 1, 2, 1) in FORTRAN—because you want to start at the beginning of each of the `time`, `lon`, and `lat` dimensions, but you want to begin at the second value of the `level` dimension. The edge lengths should be (3, 1, 5, 10) in C—or (10, 5, 1, 3) in FORTRAN—since you want to get data for all three `time` values, only one `level` value, all five `lat` values, and all 10 `lon` values. You should expect to get a total of 150 float values returned (3 * 1 * 5 * 10), and should provide enough space in your array for this many. The order in which the data will be returned is with the last dimension, `lon`, varying fastest for C, or with the first dimension, `LON`, varying fastest for FORTRAN:

```
          C                        FORTRAN

temp[0][1][0][0]         TEMP(1, 1, 2, 1)
temp[0][1][0][1]         TEMP(2, 1, 2, 1)
temp[0][1][0][2]         TEMP(3, 1, 2, 1)
temp[0][1][0][3]         TEMP(4, 1, 2, 1)

      ...                      ...

temp[2][1][4][7]         TEMP( 8, 5, 2, 3)
temp[2][1][4][8]         TEMP( 9, 5, 2, 3)
temp[2][1][4][9]         TEMP(10, 5, 2, 3)
```

Note that the different dimension orders for the C and FORTRAN interfaces do not reflect a different order for values stored on the disk, but merely different orders supported by the procedural interfaces to the two languages. In general, it does not matter whether a netCDF file is written using the C or FORTRAN interface; netCDF files written from either language may be read by programs written in the other language.

Besides the regular hyperslab access described above, the netCDF abstraction also supports the reading and writing of *generalized hyperslabs*. A generalized hyperslab has the same corner point and edge lengths attributes as a regular hyperslab. It allows, however, more general mappings between the points of the hyperslab and both the disk-resident values of the netCDF variable and the locations for those values in memory.

In a regular hyperslab, the mapping between the points of the hyperslab and the values of a netCDF variable can be described as contiguous: along each dimension, adjacent hyperslab points correspond to adjacent netCDF variable values. In a generalized hyperslab, however, this is not necessarily true; in any dimension, adjacent generalized hyperslab points correspond to netCDF variable values that are separated by a distance of n values. n is the *stride* of the dimension and needn't be the same for all dimensions. Strides may reasonably vary from one (to access adjacent netCDF variable values) to the number of points in a netCDF dimension. A regular hyperslab has unity strides in all dimensions.

The other mapping allowed by generalized hyperslabs is between the points of the hyperslab and the memory locations of the corresponding values. In a regular hyperslab, this mapping is trivial: the structure of the in-memory values (i.e. the dimensional sizes and their order) is identical to that of the hyperslab. In a generalized hyperslab, however, this is not necessarily true, instead an *index mapping vector* is used to define the mapping between points in the generalized hyperslab and the memory locations of the corresponding values. The offset, in bytes, from the origin of a memory-resident array to a particular point is given by the *inner product*[1] of the index mapping vector with the point's *coordinate offset vector*[2] . The index mapping vector for a regular hyperslab would have – in order from most rapidly varying dimension to most slowly – the byte size of a memory-resident datum (e.g. 4 for a floating-point value), then the product of that value with the edge length of the most rapidly varying dimension of the hyperslab, then the product of that value with the edge length of the next most rapidly varying dimension, and so on. In a generalized hyperslab, however, the correspondence between hyperslab points and memory locations can be radically diferent. For example, the following C definitions

```
struct vel {
    int flags;
    float u;
    float v;
} vel[NX][NY];
long imap[2] = {
    sizeof(struct vel),
    sizeof(struct vel)*NY};
```

[1] The *inner product* of two vectors [x0, x1, ..., xn] and [y0, y1, ..., yn] is just x0*y0 + x1*y1 + ... + xn*yn.

[2] A point's *coordinate offset vector* gives, for each dimension, the offset from the origin of the containing array to the point. In C, a point's coordinate offset vector is the same as it's coordinate vector. In FORTRAN, however, the values of a point's coordinate offset vector are one less than the corresponding values of the point's coordinate vector.

where `imap` is the index mapping vector, can be used to access the memory-resident values of the netCDF variable, `vel[NY][NX]`, even though the dimensions are transposed and the data is contained in a 2-D array of structures rather than a 2-D array of floating-point values.

Note that, although the netCDF abstraction allows the use of generalized hyperslab access if warranted by the situation, it does not mandate it. If you do not need generalized hyperslab access, you may ignore this capability and use regular hyperslab access instead.

To perform conventional record-oriented access, you specify a netCDF file, a record variable (one defined with an unlimited dimension), and for the record number use the value of the first dimension (last dimension in FORTRAN), using hyperslab access to get the record of values.

You may read or write multiple record variables, even if the variables are of different types, with a single call in the C interface. In this case you must write a whole record's-worth of data for each desired variable. This interface is supported as a convenience for C programmers, but is not strictly necesary, since the same result may be achieved with one read or write call for each variable. An equivalent portable FORTRAN interface that replaces multiple calls with a single call is unfortunately not possible. However, data written using the C interface can still be read with the FORTRAN interface, using one call per variable.

When efficiency is a concern, you should keep in mind the order in which netCDF data are written on the disk, since the best I/O performance is achieved by reading or writing contiguous data. All variable data are ordered with the last dimension for each variable varying fastest in the C interface, or the slowest in the FORTRAN interface. This means that for record variables in particular, at least one disk access per record will be required for reading a value from each record. Hence reading a hyperslab that takes one value out of each record will require as many disk accesses as the number of values requested. For writing, the situation is even worse, since each record must first be read and then rewritten to change a single value within a record. If you have a choice about the order in which data is accessed or the order of the dimensions that define the shape of a variable, try to choose these two orders in harmony to avoid needless inefficiency.

3.3 Data Structures

The only kind of data structure directly supported by the netCDF abstraction is a collection of named scalar and multidimensional variables with attached vector attributes. NetCDF is not particularly well-suited for storing linked lists, trees, sparse matrices, or other kinds of data structures requiring pointers. The underlying XDR library on which netCDF is implemented is quite suitable for storing and retrieving arbitrary data structures in a network-transparent way, but such structures will no longer be self-describing unless you encode information about the structure with the data. It is possible to build other kinds of data structures from sets of multidimensional arrays by adopting various conventions regarding the use of data in one array as pointers into another array. The netCDF library won't provide much help or hindrance with constructing such data structures, but netCDF provides the mechanisms with which such conventions can be designed.

For example, it is possible to use a variable attribute to name an associated index variable: the variable attribute '`array_index_var = "v_index"`' might provide the name of another associated variable to be used as an index for fast retrieval by value in the variable to which the attribute is attached.

As another example, netCDF variables may be grouped within a netCDF file by defining attributes that list the names of the variables in each group, separated by a conventional delimiter such as a space or comma. A convention can be adopted to use particular sorts of attribute names for such groupings, so that an arbitrary number of named groups of variables can be supported. If needed, a particular conventional attribute for each variable might list the names of the groups of which it is a member. Use of attributes or variables that refer to other attributes or variables provides a flexible mechanism for representing complex structures in netCDF files.

4 Use of the NetCDF Library

It is not necessary to know about the whole netCDF interface to make use of the netCDF library. If you are creating a netCDF file, only a handful of routines are required to define the necessary dimensions, variables, and attributes, and to write the data to the netCDF file. 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 file and access the data. Only authors of generic applications that access arbitrary netCDF files and write out transformed netCDF files need to be familiar with the whole netCDF library. In this chapter we provide templates of common sequences of netCDF subroutine calls needed for common uses. Full argument lists for the procedures and subroutines are described in later chapters.

4.1 Creating a NetCDF File

The typical sequences of C netCDF calls used to create a new netCDF file follows. For clarity we only present the name of the routines, omit all declarations, parameters and error checking, and use ... to represent arbitrary sequences of other statements:

```
nccreate      /* create netCDF file: enter define mode */
     ...
   ncdimdef   /* dimension definitions: from name and size */
     ...
   ncvardef   /* variable definitions: from name, type, dimensions */
     ...
   ncattput   /* attribute put: assign attribute values */
     ...
ncendef       /* end definitions: leave define mode */
     ...
ncvarput      /* variable put: provide values for variables */
  ...
ncattput      /* attribute put: change attribute values */
  ...
ncclose       /* close: save new netCDF file */
```

In FORTRAN, the corresponding sequence looks like this:

```
NCCRE             ! create netCDF file: enter define mode
     ...
   NCDDEF         ! define dimensions: from name and size
```

```
          ...
       NCVDEF           ! define variables: from name, type, dimensions
          ...
       NCAPT or NCAPTC ! attribute put: assign attribute values
          ...
    NCENDF              ! end definitions: leave define mode
          ...
    NCVPT or NCVPTC     ! variable put: provide values for variables
      ...
    NCAPT or NCAPTC     ! attribute put: change attribute values
      ...
    NCCLOS              ! close: save new netCDF file
```

The FORTRAN interface provides two subroutines for defining attributes and providing values for variables, depending on whether a numeric or character string value is used. The FORTRAN template indicates that either of these subroutines could be called.

Only one call is needed to begin creating a netCDF file, at which point you will be in the first of two netCDF *modes*. When accessing a netCDF, you are 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 `ncdimdef` (or `NCDDEF`) is needed for each dimension created. Similarly, one call to `ncvardef` (or `NCVDEF`) is needed for each variable creation, and one call to `ncattput` (or `NCAPT` or `NCAPTC`) is needed for each attribute defined and assigned a value. The only way to leave define mode and enter data mode is by a call to `ncendef` (or `NCENDF`).

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 for the attribute). Single values are written to a variable with `ncvarput1` (or `NCVPT1` or `NCVP1C`); while arbitrary hyperslabs of data are written using `ncvarput` or `ncvarputg` (or `NCVPT`, `NCVPTC`, `NCVPTG`, or `NCVPGC`) instead. Multi-variable records of data may be written using multiple calls to `ncvarput` (or `NCVPT`) or with a single call to `ncrecput`.

Finally, you should explicitly close all open netCDF files on which you are writing by calling `ncclose` (or `NCCLOS`) before the program exits. If a program terminates abnormally with netCDF files open for writing, you may lose one or more records of the most recently written record variable data as well as any attribute changes since the last call to `ncsync` (or `NCSNC`). It is possible to reduce the chance of losing data due to abnormal termination by explicitly calling `ncsync` (`NCSNC`) after every write to netCDF variables or change to attribute values. This can be expensive in computer resources, so such calls should ordinarily be omitted unless they are really needed.

4.2 Reading a NetCDF File with Known Names

If you know the names of the dimensions, variables, and attributes in a netCDF file, you can write calls to read data from the file; you don't need to include the "inquire" calls that determine the dimensions, variables, and attributes. If you employ such knowledge about particular netCDF files, the program you write will lack generality. It will only work with files that have the assumed names and structure, so you will be losing some of the advantages of using the netCDF interface. However, you may be writing software that expects the user or some other program to supply variable or dimension names, perhaps as subroutine or command line arguments. In that case, the resulting program could be quite general.

When you know the names of some variables of interest and their dimensions, the order of typical C calls to read data from those variables in a netCDF file is:

```
ncopen          /* open existing netCDF */
  ...
ncdimid         /* get dimension IDs to use in accessing data */
  ...
ncvarid         /* get variable IDs */
  ...
ncattget        /* get attribute values, if needed */
  ...
ncvarget        /* get values of variables */
  ...
ncclose         /* close netCDF */
```

In FORTRAN, the corresponding sequence looks like this:

```
NCOPN           !  open existing netCDF
  ...
NCDID           !  get dimension IDs to use in accessing data
  ...
NCVID           !  get variable IDs
  ...
NCAGT or NCAGTC !  get attribute values, if needed
  ...
```

```
NCVGT or NCVGTC  !  get values of variables
  ...
NCCLOS           !  close netCDF
```

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

Next, a call to `ncdimid` (or `NCDID`) for each dimension of interest gets the dimension ID from the dimension name. Dimension IDs, like netCDF IDs, are small integers used to refer to dimensions in subsequent calls. Similarly, each required variable ID is determined from its name by a call to `ncvarid` (or `NCVID`). 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 `ncattget` (or `NCAGT` or `NCAGTC`) for each desired attribute. Variable data values can be directly accessed from the netCDF file with `ncvarget1` (or `NCVGT1` or `NCVG1C`) for single values, `ncvarget` or `ncvargetg` (or `NCVGT`, `NCVGTC`, `NCVGTG`, or `NCVGGC`) for hyperslabs of values, or `ncrecget` for records of values. To minimize the number of disk accesses, you should remember that the last dimension in C (first dimension in FORTRAN) varies fastest when using hyperslab access.

Finally, the netCDF file can be closed with `ncclose` (or `NCCLOS`) when you are finished with it to free system resources. There is no harm in not closing a file open only for reading.

4.3 Reading a netCDF File with Unknown Names

If you want to write generic software (i.e., a program that transposes specified variables by interchanging specified dimensions) you should make no assumptions about the dimension and variable names that are not specified. In such cases, you must find out about all the dimensions, variables, and attributes in a netCDF file by calling the inquire functions. Four inquire functions get information about a whole netCDF file, a dimension, a variable, or an attribute. The following template illustrates how they are used:

```
ncopen          /* open existing netCDF */
  ...
ncinquire       /* find out what is in it */
     ...
   ncdiminq     /* get dimension names, sizes */
     ...
   ncvarinq     /* get variable names, types, shapes */
        ...
      ncattname /* get attribute names */
        ...
      ncattinq  /* get attribute types and lengths */
        ...
      ncattget  /* get attribute values */
        ...
ncvarget        /* get values of variables */
  ...
ncclose         /* close netCDF */
```

In FORTRAN, the corresponding sequence looks like this:

```
NCOPN                !  open existing netCDF
  ...
NCINQ                !  find out what is in it
     ...
   NCDINQ            !  get dimension names, sizes
     ...
   NCVINQ            !  get variable names, types, shapes
        ...
      NCANAM         !  get attribute names
        ...
      NCAINQ         !  get attribute values
        ...
      NCAGT or NCAGTC !  get attribute values
        ...
NCVGT or NCVGTC      !  get values of variables
  ...
NCCLOS               !  close netCDF
```

As in the previous example, a single call opens the existing netCDF file, returning a netCDF ID. This netCDF ID is given to the `ncinquire` (or `NCINQ`) 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.

Another inquire function, `ncrecinq`, is provided for convenience to return information about record variables, although the information is also obtainable by using the other inquire functions. The `ncrecinq` function returns the number of record variables, their variable IDs, and how much memory space is needed for a record's worth of data for each record variable.

All the inquire functions are quite inexpensive to use and require no I/O, since the information they provide is stored in a table in memory for each open netCDF file. In the C interface, the inquire functions also support getting a subset of information by providing NULL pointers instead of valid addresses for undesired information.

Dimension IDs are assigned by using consecutive integers (beginning at 0 in C, 1 in FORTRAN). Also dimensions, once created, cannot be deleted. Therefore, knowing the number of dimension IDs in a netCDF file means knowing all the dimension IDs: they are the integers 0, 1, 2, ..., (or 1, 2, 3, ... in FORTRAN). For each dimension ID, a call to the inquire function `ncdiminq` (or `NCDINQ`) returns the dimension name and size.

Like dimension IDs, variable IDs are also 0, 1, 2, ..., (or 1, 2, 3, ... in FORTRAN). These can be used in `ncvarinq` (or `NCVINQ`) 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 `ncattname` (or `NCANAM`) return the name for each attribute given the netCDF ID, variable ID, and attribute number. Armed with the attribute name, a call to `ncattinq` (or `NCAINQ`) returns its type and length. Given the type and length, the generic application can allocate enough space to hold the attribute values. Then a call to `ncattget` (or `NCAGT` or `NCAGTC`) returns the attribute values.

Once the names, IDs, types, shapes, and lengths of all netCDF components are known, data values can be accessed by calling `ncvarget1` (or `NCVGT1` or `NCVG1C`) for single values, `ncvarget` or `ncvargetg` (or `NCVGT`, `NCVGTC`, `NCVGTG`, or `NCVGGC`) for aggregates of values using hyperslab access, or `ncrecget` for aggregates of values using record access.

4.4 Adding New Dimensions, Variables, Attributes

An existing netCDF file 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 file:

```
ncopen          /* open existing netCDF file */
  ...
ncredef         /* put it into define mode */
    ...
  ncdimdef      /* define additional dimensions (if any) */
    ...
  ncvardef      /* define additional variables (if any) */
    ...
  ncattput      /* define additional attributes (if any) */
    ...
ncendef         /* check all definitions, leave define mode */
  ...
ncvarput        /* provide values for new variables */
  ...
ncclose         /* save netCDF file */
```

In FORTRAN, the corresponding sequence looks like this:

```
NCOPN             !  open existing netCDF
  ...
NCREDF            !  put it into define mode
    ...
  NCDDEF          !  define additional dimensions (if any)
    ...
  NCVDEF          !  define additional variables (if any)
    ...
  NCAPT or NCAPTC !  define additional attributes (if any)
    ...
NCENDF            !  check all definitions, leave define mode
  ...
NCVPT or NCVPTC   !  provide values for new variables
  ...
NCCLOS            !  save netCDF file
```

A netCDF file is first opened by the `ncopen` (or `NCOPN`) call. This call puts you 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 leave data mode and enter *define mode*, by calling `ncredef` (or `NCREDF`). In define mode, call `ncdimdef` (or `NCDDEF`) to define new dimensions, `ncvardef` (or `NCVDEF`) to define new variables (using the new dimensions), and `ncattput` (or `NCAPT` or `NCAPTC`) 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 `ncendef` (or `NCENDF`). If you do not wish to reenter data mode, just call `ncclose` (or `NCCLOS`), which will have the effect of first calling `ncendef` (or `NCENDF`).

Until the `ncendef` (or `NCENDF`) call, you may back out of all the redefinitions made in define mode and restore the previous state of the netCDF by calling `ncabort` (or `NCABOR`). You may also use the `ncabort` call to restore the netCDF to a consistent state if the call to `ncendef` (or `NCENDF`) fails. If you have called `ncclose` (or `NCCLOS`) from definition mode and the implied call to `ncendef` (or `NCENDF`) fails, `ncabort` (or `NCABOR`) will automatically be called to close the netCDF in its previous consistent state (before you entered define mode).

4.5 Error Handling

By default all netCDF library routines print an error message and exit when an error has occurred. If this error behavior is acceptable, you never need to check error returns, since any condition that would result in an error will print an explanatory message and exit. The examples in this guide assume the default error-handling behavior, so there is no checking of error returns.

Occasionally, low-level write errors may occur in the XDR library 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 message from one of the XDR functions rather than from the netCDF library. If you get a message from the XDR layer, diagnose and correct whatever is causing the low-level write errors.

In the C interface, errors may be handled more flexibly by setting the external integer `ncopts`, declared in the file '`netcdf.h`'. Two aspects of the error-handling behavior can be modified independently: the suppression of error messages, and the fatality of errors. The default behavior is specified by the assignment

```
ncopts = NC_VERBOSE | NC_FATAL;
```

where `NC_VERBOSE` and `NC_FATAL` are predefined constants from the include file '`netcdf.h`'.

If you want error messages but do not wish errors to be fatal, turn off the fatal error flag with:

```
ncopts = NC_VERBOSE;
```

If you want neither error messages nor fatal errors, turn off both flags with:

```
ncopts = 0;
```

In either case, you should check the return value after each call to a netCDF function. The integer `-1` is returned whenever an error occurs and `NC_FATAL` is off, so you can detect error returns and handle the errors appropriately. Another externally-defined integer, `ncerr`, contains a netCDF-specific error code that can be used after an error has occurred to determine what the nature of the error was. The names and descriptions of netCDF error codes are included in the file '`netcdf.h`'.

In the FORTRAN interface, the error options described above can be accessed by using the routines NCPOPT and NCGOPT. The default error- handling behavior is equivalent to the statement

```
      CALL NCPOPT(NCVERBOS+NCFATAL)
```

where the values of NCVERBOS and NCFATAL are pre-defined constants from the FORTRAN include file '`netcdf.inc`'. If you want error messages, but do not wish errors to be fatal, turn off the fatal error flag with:

```
      CALL NCPOPT(NCVERBOS)
```

If you want neither error messages nor fatal errors, turn off both flags with:

```
      CALL NCPOPT(0)
```

To get the current value of the error options, use:

```
      CALL NCGOPT(NCOPTS)
```

In either case, the integer return code (the last parameter in all of the FORTRAN subroutines and functions) contains the non-zero netCDF-specific error number that can be used to determine the nature of the error. Names and descriptions of netCDF error codes are included in the file '`netcdf.inc`'.

5 NetCDF Operations

This chapter presents the interfaces of the netCDF routines that deal with a netCDF file as a whole.

A netCDF file that has not yet been opened can only be referred to by its file name. Once a netCDF file is opened, it is referred to by an ID, which is a small nonnegative integer returned when you create or open the file. A netCDF ID is a file *handle*, much like a file descriptor in C or a logical unit number in FORTRAN. In any single program, the netCDF IDs of distinct open netCDFs are distinct. A single netCDF file 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 file should permit writing. When an open netCDF file is closed, its ID no longer refers to it, and that ID may be subsequently reassigned to refer to a different netCDF that is opened later.

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

- Create, given file path and whether to overwrite or not.
- Open for access, given file path 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.
- Get 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 C and FORTRAN interfaces, the rest of this chapter presents the interfaces for these operations.

5.1 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 list of possible error conditions;
- A C 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;
- An example of a C program fragment calling the netCDF function and perhaps other netCDF functions to do something useful;
- A FORTRAN function prototype that presents the type and order of the formal parameters to the FORTRAN function or functions that provide the same functionality as the C function;
- A description of each formal parameter in the FORTRAN interface; and
- An example of a FORTRAN program fragment that duplicates the function of the example C fragment.

The C function prototypes specify the order and type of each formal parameter and conform to the ANSI C standard. FORTRAN does not have function prototypes, but a similar syntax is used to concisely present the order and types of FORTRAN formal parameters. In the few cases in which a single C function corresponds to two FORTRAN functions, the FORTRAN functions prototypes are presented together.

The FORTRAN examples use two nonstandard notations: `INCLUDE` statements and in-line comments. In each case, we use the VMS FORTRAN notation, as in the following example:

```
INCLUDE 'netcdf.inc'
INTEGER NCID       ! this is an in-line comment
```

FORTRAN examples (and the FORTRAN interface) abide by the six-character limitation on the length of FORTRAN names, except that parameters names may be up to eight characters long.

5.2 Create a NetCDF file

The function `nccreate` (or `NCCRE` for FORTRAN) creates a new netCDF file, returning a netCDF ID that can subsequently be used to refer to the netCDF file. The new netCDF file is placed in define mode.

In case of an error, `nccreate` returns -1; `NCCRE` returns a nonzero value in `rcode`. Possible causes of errors include::

- Passing a file name that includes a directory that does not exist.
- Specifying a file name of a file that exists and also specifying `NC_NOCLOBBER` (or `NCNOCLOB`).
- Attempting to create a netCDF file in a directory where you don't have permission to create files.

nccreate: C Interface

```
int nccreate (const char* path, int cmode);
```

`path` The file name of the new netCDF file. This can be given as either an absolute path name (from the root of the file system) or a relative path name (from the current directory).

`cmode` Should be specified as either `NC_CLOBBER` or `NC_NOCLOBBER`. These constants are defined in the include file named '`netcdf.h`'. `NC_CLOBBER` means that even if the file already exists, you want to create a new file with the same name, erasing the old file's contents. `NC_NOCLOBBER` means you want to create a new netCDF file only if the given file name does not refer to a file that already exists.

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

```
#include "netcdf.h"
   ...
int ncid;
   ...
ncid = nccreate("foo.nc", NC_NOCLOBBER);
```

NCCRE: FORTRAN Interface

```
INTEGER FUNCTION NCCRE (CHARACTER*(*) PATH, INTEGER CMODE,
                           INTEGER RCODE)
```

`PATH` The file name of the new netCDF file. This can be given as either an absolute path name (from the root of the file system) or a relative path name (from the current directory).

`CMODE` Should be specified as either `NCCLOB` or `NCNOCLOB`. These constants are defined in the include file '`netcdf.inc`'. `NCCLOB` means that even if the file already exists, you want to create a new file with the same name, erasing the old file's contents. `NCNOCLOB` means you want to create a new netCDF file only if the given file name does not refer to a file that already exists.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

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

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)
```

5.3 Open a NetCDF File for Access

The function `ncopen` (or `NCOPN` for FORTRAN) opens an existing netCDF file for access.

In case of an error, `ncopen` returns -1; `NCOPN` returns a nonzero value in `rcode`. Possible causes of errors include::

- The specified netCDF file does not exist.
- The mode specified is something other than `NC_WRITE` or `NC_NOWRITE`.

ncopen: C Interface

```
int ncopen(const char* path,int mode);
```

`path` Absolute or relative file name for netCDF file to be opened.

`mode` Either `NC_WRITE`, to open the file for writing, or `NC_NOWRITE`, to open the file read-only. "Writing" means any kind of change to the file, including appending or changing data, adding or renaming dimensions, variables, and attributes, or deleting attributes.

Here is an example using `ncopen` to open an existing netCDF file named '`foo.nc`' for reading:

```
#include "netcdf.h"
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
```

NCOPN: FORTRAN Interface

```
      INTEGER FUNCTION NCOPN(CHARACTER*(*) PATH,
     +                       INTEGER RWMODE,
     +                       INTEGER RCODE)
```

`PATH` Absolute or relative file name for netCDF file to be opened.

`RWMODE` Either `NCWRITE`, to open the file for writing, or `NCNOWRIT`, to open the file read-only. "Writing" means any kind of change to the file, including appending or changing data, adding or renaming dimensions, variables, and attributes, or deleting attributes.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example of using `NCOPN` to open an existing netCDF file named '`foo.nc`' for reading:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
```

5.4 Put Open NetCDF File into Define Mode

The function `ncredef` (or `NCREDF` for FORTRAN) puts an open netCDF file into define mode, so dimensions, variables, and attributes can be added or renamed and attributes can be deleted.

In case of an error, `ncredef` returns -1; `NCREDF` returns a nonzero value in `rcode`. Possible causes of errors include::

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

ncredef: C Interface

```
int ncredef(int ncid);
```

`ncid` netCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncredef` to open an existing NetCDF file named '`foo.nc`' and put it into define mode:

```
#include "netcdf.h"
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);    /* open file */
   ...
ncredef(ncid);                        /* put in define mode */
```

NCREDF: FORTRAN Interface

```
SUBROUTINE NCREDF(INTEGER NCID, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example of using `NCREDF` to open an existing netCDF file named '`foo.nc`' and put it into define mode:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCOPN('foo.nc', NCWRITE, RCODE)
   ...
CALL NCREDF(NCID, RCODE)
```

5.5 Leave Define Mode

The function `ncendef` (or `NCENDF` for FORTRAN) takes an open netCDF file out of define mode. The changes made to the netCDF file while it was in define mode are checked and committed to disk if no problems occurred. The netCDF file is then placed in data mode, so variable data can be read or written.

This call can be expensive, since it involves initializing non-record variables and copying data under some circumstances. See ⟨undefined⟩ [NetCDF File Structure and Performance], page ⟨undefined⟩, for a more extensive discussion.

In case of an error, `ncendef` returns -1; `NCENDF` returns a nonzero value in `rcode`. Possible causes of errors include:

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

ncendef: C Interface

```
int ncendef(int ncid);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

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

```
#include "netcdf.h"
   ...
int ncid;
   ...
ncid = nccreate("foo.nc", NC_NOCLOBBER);

   ...      /* create dimensions, variables, attributes */

ncendef(ncid);  /*leave define mode*/
```

NCENDF: FORTRAN Interface

```
SUBROUTINE NCENDF(INTEGER NCID, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

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

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)

   ...  !  create dimensions, variables, attributes

CALL NCENDF(NCID, RCODE)
```

5.6 Close an Open NetCDF File

The function `ncclose` (or `NCCLOS` for FORTRAN) closes an open netCDF file. If the file is in define mode, `ncendef` (or `NCENDF`) will be called before closing. (In this case, if `ncendef` [or `NCENDF`] returns an error, `ncabort` [or `NCABOR`] will automatically be called to restore the file to the consistent state before define mode was last entered.) After an open netCDF file is closed, its netCDF ID will be reassigned to the next netCDF file that is opened or created.

In case of an error, `ncclose` returns -1; `NCCLOS` returns a nonzero value in `rcode`. Possible causes of errors include:

- The netCDF was in define mode and the automatic call made to `ncendef` (or `NCENDF`) failed.
- The specified netCDF ID does not refer to an open netCDF file.

ncclose: C Interface

```
int ncclose(int ncid);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncclose` to finish the definitions of a new netCDF file named '`foo.nc`' and release its netCDF ID:

```
#include "netcdf.h"
   ...
int ncid;
   ...
ncid = nccreate("foo.nc", NC_NOCLOBBER);

   ...      /* create dimensions, variables, attributes */

ncclose(ncid);       /* close netCDF file */
```

NCCLOS: FORTRAN Interface

```
SUBROUTINE NCCLOS(INTEGER NCID, INTEGER RCODE)
```

`NCID` netCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCCLOS` to finish the definitions of a new netCDF file named '`foo.nc`' and release its netCDF ID:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, RCODE
   ...
NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)

   ...  ! create dimensions, variables, attributes

CALL NCCLOS(NCID, RCODE)
```

5.7 Inquire about an Open NetCDF File

The function `ncinquire` (`NCINQ` for FORTRAN) returns information about an open netCDF file, given its netCDF ID. It can be called from either define mode or data mode. It 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 size, if any. No I/O is required when this or any other 'inquire' function in the netCDF interface is called, since the functions merely return information that is stored in a table for each open netCDF file.

In case of an error, `ncinquire` returns -1; `NCINQ` returns a nonzero value in `rcode`. Possible cause of errors includes:

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

ncinquire: C Interface

```
int ncinquire(int ncid, int* ndims, int* nvars, int* ngatts,
              int* recdim);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`ndims` Returned number of dimensions defined for this netCDF file. If this parameter is given as '`(int *) 0`', the number of dimensions will not be returned so no variable to hold this information needs to be declared.

`nvars` Returned number of variables defined for this netCDF file. If this parameter is given as '`(int *) 0`', the number of variables will not be returned so no variable to hold this information needs to be declared.

`ngatts` Returned number of global attributes defined for this netCDF file. If this parameter is given as '`(int *) 0`', the number of global attributes will not be returned so no variable to hold this information needs to be declared.

`recdim` Returned ID of the unlimited dimension, if there is one for this netCDF file. If no unlimited size dimension has been defined, -1 is returned for the value of `recdim`. If this parameter is given as '`(int *) 0`', the record dimension ID will not be returned so no variable to hold this information needs to be declared.

Here is an example using `ncinquire` to find out about a netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int ncid, ndims, nvars, ngatts, recdim;
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
ncinquire(ncid, &ndims, &nvars, &ngatts, &recdim);
```

NCINQ: FORTRAN Interface

```
      SUBROUTINE NCINQ(INTEGER NCID, INTEGER NDIMS, INTEGER NVARS,
     *                 INTEGER NGATTS, INTEGER RECDIM, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`NDIMS` Returned number of dimensions defined for this netCDF file.

`NVARS` Returned number of variables defined for this netCDF file.

`NGATTS` Returned number of global attributes defined for this netCDF file.

`RECDIM` Returned ID of the unlimited dimension, if there is one for this netCDF file. If no unlimited size dimension has been defined, -1 is returned for the value of `RECDIM`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCINQ` to find out about a netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, NDIMS, NVARS, NATTS, RECDIM, RCODE
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
         ...
      CALL NCINQ(NCID, NDIMS, NVARS, NATTS, RECDIM, RCODE)
```

5.8 Synchronize an Open NetCDF File to Disk

The function `ncsync` (or `NCSNC` for FORTRAN) makes sure that the disk copy of a netCDF file open for writing is current. The netCDF file must be in data mode. A netCDF file in define mode is synchronized to disk only when `ncendef` (or `NCENDF`) is called. A process that is reading a netCDF file that another process is writing can also call `ncsync` (or `NCSNC` for FORTRAN) to get updated with the changes made by the writing process (e.g. the number of records written), without having to close and reopen the file.

It can be expensive in computer resources to always synchronize to disk after every write of variable data or change of an attribute value. 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 file open for reading would not see the number of records increase when the writing process calls `ncsync`; to accomplish this, the reading process must call `ncsync`.

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

In case of an error, `ncsync` returns -1; `NCSNC` returns a nonzero value in `rcode`. Possible causes of errors include:

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

ncsync: C Interface

```
int ncsync(int ncid);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncsync` to synchronize the disk writes of a netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);  /* open for writing */

   ...          /* write data or change attributes */

ncsync(ncid);      /* synchronize to disk */
```

NCSNC: FORTRAN Interface

```
      SUBROUTINE NCSNC(INTEGER NCID, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCSNC` to synchronize the disk writes of a netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
* write data or change attributes
         ...
      CALL NCSNC(NCID, RCODE)
```

5.9 Back Out of Recent Definitions

The function `ncabort` (or `NCABOR` for FORTRAN), if not in define mode, closes the netCDF file. If the file is being created and is still in define mode, the file is deleted. If define mode was entered by a call to `ncredef` (or `NCREDF`), the netCDF file is restored to its state before definition mode was entered and the file is closed. The main reason for calling `ncabort` (or `NCABOR`) is to restore the netCDF to a known consistent state in case anything goes wrong during the definition of new dimensions, variables, or attributes.

This function is called automatically if `ncclose` (or `NCCLOS`) is called from define mode and the call to leave define mode before closing fails.

In case of an error, `ncabort` returns -1; `NCABOR` returns a nonzero value in `rcode`. Possible causes of errors include:

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

ncabort: C Interface

```
int ncabort(int ncid);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

Here is an example using `ncabort` to back out of redefinitions of a file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);  /* open for writing */
   ...
ncredef(ncid);                     /* enter define mode */
   ...
if (ncdimdef(ncid, "lat", 18L) == -1)
   ncabort(ncid);                  /* define failed, abort */
```

NCABOR: FORTRAN Interface

```
SUBROUTINE NCABOR(INTEGER NCID, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCABOR` to back out of redefinitions of a file named '`foo.nc`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER NCID, RCODE, LATID
   ...
NCID = NCOPN('foo.nc', NCWRITE, RCODE)
   ...
CALL NCREDF(NCID, RCODE)
   ...
LATID = NCDDEF(NCID, 'LAT', 18, RCODE)
IF (RCODE .EQ. -1) THEN  ! dimension definition failed
   CALL NCABOR(NCID, RCODE)  ! abort redefinitions
ENDIF
   ...
```

5.10 Set Fill Mode for Writes

These calls are intended for advanced usage, to optimize writes under some circumstances described below. The function `ncsetfill` (or `NCSFIL` for FORTRAN) sets the *fill mode* for an netCDF file open for writing and returns the current fill mode. The fill mode can be specified as either `NC_FILL` or `NC_NOFILL` (`NCFILL` or `NCNOFILL` for FORTRAN). The default behavior corresponding to `NC_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 hasn't been written yet. This makes it possible to detect attempts to read data before it was written. See ⟨undefined⟩ [Missing Values], page ⟨undefined⟩, for more information on the use of fill values. See ⟨undefined⟩ [Attribute Conventions], page ⟨undefined⟩ for information about how to define your own fill values.

The behavior corresponding to `NC_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 immediately overwritten with data.

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

After you turn on `NC_NOFILL` mode for an open netCDF file, you must be certain to write valid data in all the positions that will later be read. Note that `NC_NOFILL` mode is only a transient property of a netCDF file open for writing: if you close and reopen the file, it will revert to the default behavior. You can also revert to the default behavior by calling `ncsetfill` (or `NCSFIL` for FORTRAN) again to explicitly set the fill mode to `NC_FILL`.

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

1. Creating and initializing a netCDF file. In this case, you should set nofill mode before calling `ncendef` (`NCENDF` for FORTRAN), 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 file. Set nofill mode after opening the file for writing, then append the additional records to the file completely, leaving no intervening unwritten records.
3. Adding new variables that you are going to initialize to an existing netCDF file. Set nofill mode before calling `ncendef` (`NCENDF` for FORTRAN), then write all the new variables completely.

If the netCDF file has an unlimited dimension and the last record was written while in `NC_NOFILL` mode, then the file will be 4 bytes longer than if `NC_NOFILL` mode wasn't set, but this will be completely transparent if you access the data only through the netCDF interfaces.

In case of an error, `ncsetfill` returns -1; `NCSFIL` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified netCDF ID does not refer to an open netCDF file.
- The specified netCDF ID refers to a file open for read-only access.
- The fillmode argument is neither `NC_NOFILL` nor `NC_FILL` (neither `NCNOFILL` nor `NCFILL` for FORTRAN).

ncsetfill: C Interface

```
int ncsetfill(int ncid, int fillmode);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`fillmode` Desired fill mode for the file, either `NC_NOFILL` or `NC_FILL`.

`ncsetfill`
: Returns the current fill mode of the file before this call, either `NC_NOFILL` or `NC_FILL`.

Here is an example using `ncsetfill` to set nofill mode for subsequent writes of a netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int ncid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);  /* open for writing */

   ...         /* write data with default prefilling behavior */

ncsetfill(ncid, NC_NOFILL);      /* set nofill mode */

   ...         /* write data with no prefilling */
```

NCSFIL: FORTRAN Interface

```
      INTEGER FUNCTION NCSFIL(INTEGER NCID, INTEGER FILLMODE,
     +                        INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`FILLMODE` Desired fill mode for the file, either `NCNOFILL` or `NCFILL`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

`NCSFIL` Returns the current fill mode of the file before this call, either `NCNOFILL` or `NCFILL`.

Here is an example using `NCSFIL` to set nofill mode for a netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, OMODE
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
* write data with default prefilling behavior
         ...
      OMODE = NCSFIL(NCID, NCNOFILL, RCODE)
         ...
* write data with no prefilling
         ...
```

6 Dimensions

Dimensions for a netCDF file are defined when it is created, while the netCDF file is in define mode. Additional dimensions may be added later by reentering define mode. A netCDF dimension has a name and a size. At most one dimension in a netCDF file can have the `NC_UNLIMITED` size, which means a variable using this dimension can grow to any length (like a record number in a file).

There is a suggested limit (currently 32) to the number of dimensions that can be defined in a single netCDF file. The limit is the value of the predefined macro `MAX_NC_DIMS` (MAXNCDIM for FORTRAN). The purpose of the limit is to make writing generic applications simpler. They need only provide an array of `MAX_NC_DIMS` dimensions to handle any netCDF file. The implementation of the netCDF library does not enforce this advisory maximium, so it is possible to use more dimensions, if necessary; just don't expect generic applications or netCDF utilities to be able to handle the resulting netCDF files.

Ordinarily, the name and size of a dimension are fixed when the dimension is first defined. The name may be changed later, but the size of a dimension cannot be changed without copying the netCDF to a new netCDF with a redefined dimension size.

Dimension sizes in the C interface are type `long` rather than type `int` to make it possible to access all the data in a netCDF file on a platform that only supports a 16-bit `int` data types, for example MSDOS. If dimension sizes were type `int` instead, it would not be possible to access data from variables with a dimension size greater than a 16-bit `int` can accommodate.

Operations supported on dimensions are:

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

6.1 Create a Dimension

The function `ncdimdef` (or `NCDDEF` for FORTRAN) adds a new dimension to an open netCDF file in define mode. It returns a dimension ID, given the netCDF ID, the dimension name, and the dimension size. At most one unlimited size dimension, called the record dimension, may be defined for each netCDF file.

In case of an error, `ncdimdef` returns -1; `NCDDEF` returns a nonzero value in `rcode`. Possible causes of errors include:

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

ncdimdef: C Interface

```
int ncdimdef(int ncid, const char* name, long size);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`name` Dimension name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant.

`size` Size 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 (of type `long`) or the predefined constant `NC_UNLIMITED`.

Here is an example using `ncdimdef` to create a dimension named `lat` of size 18 and a record dimension named `rec` in a new netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int ncid, latid, recid;
   ...
ncid = nccreate("foo.nc", NC_NOCLOBBER);
   ...
latid = ncdimdef(ncid, "lat", 18L);
recid = ncdimdef(ncid, "rec", NC_UNLIMITED);
```

NCDDEF: FORTRAN Interface

```
      INTEGER FUNCTION NCDDEF (INTEGER NCID,
     +                    CHARACTER*(*) DIMNAM,
     +                    INTEGER DIMSIZ,
     +                    INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`DIMNAM` Dimension name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant.

`DIMSIZ` Size 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 `NCUNLIM`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCDDEF` to create a dimension named `lat` of size 18 and a record dimension named `rec` in a new netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID, RECID
         ...
      NCID = NCCRE('foo.nc', NCNOCLOB, RCODE)
         ...
      LATID = NCDDEF(NCID, 'lat', 18, RCODE)
      RECID = NCDDEF(NCID, 'rec', NCUNLIM, RCODE)
```

6.2 Get a Dimension ID from Its Name

The function `ncdimid` (or `NCDID` for FORTRAN) returns the ID of a netCDF dimension, given the name of the dimension. If `ndims` is the number of dimensions defined for a netCDF file, each dimension has an ID between `0` and `ndims-1` (or `1` and `ndims` for FORTRAN).

In case of an error, `ncdimid` returns -1; `NCDID` returns a nonzero value in `rcode`. Possible causes of errors include:

- The name that was specified is not the name of any currently defined dimension in the netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

ncdimid: C Interface

```
int ncdimid(int ncid, const char* name);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`name` Dimension name, a character string beginning with a letter and followed by any sequence of letters, digits, or underscore ('`_`') characters. Case is significant in dimension names.

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

```
#include "netcdf.h"
   ...
int ncid, latid;
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);  /* open for reading */
   ...
latid = ncdimid(ncid, "lat");
```

NCDID: FORTRAN Interface

```
      INTEGER FUNCTION NCDID (INTEGER NCID,
     +                        CHARACTER*(*) DIMNAME,
     +                        INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`DIMNAME` Dimension name, a character string beginning with a letter and followed by any sequence of letters, digits, or underscore ('`_`') characters. Case is significant in dimension names.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

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

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
         ...
      LATID = NCDID(NCID, 'lat', RCODE)
```

6.3 Inquire about a Dimension

The function `ncdiminq` (or `NCDINQ` for FORTRAN) returns the name and size of a dimension, given its ID. The size for the unlimited dimension, if any, is the maximum value used so far in writing data for that dimension (which is the same as the current maximum record number).

In case of an error, `ncdiminq` returns -1; `NCDINQ` returns a nonzero value in `rcode`. Possible causes of errors include:

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

ncdiminq: C Interface

```
int ncdiminq(int ncid, int dimid, char* name, long* size);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`dimid` Dimension ID, as returned from a previous call to `ncdimid` or `ncdimdef`.

`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 `MAX_NC_NAME`. If the name parameter is given as '`(char *) 0`', no name will be returned so no space needs to be allocated.

`size` Returned size 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. If this parameter is given as '`(long *) 0`', the size will not be returned, so no space for this information need be declared or allocated.

Here is an example using `ncdiminq` to determine the size of a dimension named `lat`, and the name and current maximum size of the unlimited (or record) dimension for an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int ncid, latid, ndims, nvars, ngatts, recid;
long latsize, recs;
char recname[MAX_NC_NAME];
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);  /* open for reading */
   ...
latid = ncdimid(ncid, "lat");
/* get lat size, but don't get name, since we already know it */
ncdiminq(ncid, latid, (char *) 0, &latsize);
/* get ID of record dimension (among other things) */
ncinquire(ncid, &ndims, &nvars, &ngatts, &recid);
/* get record dimension name and current size */
ncdiminq(ncid, recid, recname, &recs);
```

NCDINQ: FORTRAN Interface

```
      SUBROUTINE NCDINQ (INTEGER NCID, INTEGER DIMID,
     +                   CHARACTER*(*) DIMNAM, INTEGER DIMSIZ,
     +                   INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`DIMID` Dimension ID, as returned from a previous call to `NCDID` or `NCDDEF`.

`DIMNAM` 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 `MAXNCNAM`.

`DIMSIZ` Returned size 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.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCDINQ` to determine the size of a dimension named `lat`, and the name and current maximum size of the unlimited (or record) dimension for an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID, LATSIZ
      INTEGER NDIMS, NVARS, NGATTS, RECID, NRECS
* 31 in following statement is parameter MAXNCNAM
      CHARACTER*31 LATNAM, RECNAM
         ...
      NCID = NCOPN('foo.nc', NCNOWRIT, RCODE)
         ...
      LATID = NCDID(NCID, 'lat', RCODE)
* get lat name and size, (even though we already know name)
      CALL NCDINQ(NCID, LATID, LATNAM, LATSIZ, RCODE)
* get ID of record dimension (among other things)
      CALL NCINQ(NCID, NDIMS, NVARS, NGATTS, RECID, RCODE)
* get record dimension name and current size
      CALL NCDINQ(NCID, RECID, RECNAME, NRECS, RCODE)
```

6.4 Rename a Dimension

The function `ncdimrename` (or `NCDREN` for FORTRAN) renames an existing dimension in a netCDF file open for writing. If the new name is longer than the old name, the netCDF must be in define mode. You cannot rename a dimension to have the same name as another dimension.

In case of an error, `ncdimrename` returns -1; `NCDREN` returns a nonzero value in `rcode`. Possible causes of errors include:

- The new name is the name of another dimension.
- The dimension ID is invalid for the specified netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

ncdimrename: C Interface

```
int ncdimrename(int ncid, int dimid, const char* name);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`dimid` Dimension ID, as returned from a previous call to `ncdimid` or `ncdimdef`.

`name` New dimension name.

Here is an example using `ncdimrename` to rename the dimension `lat` to `latitude` in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int ncid, latid;
   ...
ncid = ncopen("foo.nc", NC_WRITE);  /* open for writing */
   ...
ncredef(ncid);  /* put in define mode to rename dimension */
latid = ncdimid(ncid, "lat");
ncdimrename(ncid, latid, "latitude");
ncendef(ncid);  /* leave define mode */
```

NCDREN: FORTRAN Interface

```
      SUBROUTINE NCDREN (INTEGER NCID, INTEGER DIMID,
     +                   CHARACTER*(*) DIMNAME, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`DIMID` Dimension ID, as returned from a previous call to `NCDID` or `NCDDEF`.

`DIMNAM` New name for the dimension.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCDREN` to rename the dimension "lat" to "latitude" in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER NCID, RCODE, LATID
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
         ...
* put in define mode to rename dimension
      CALL NCREDF(NCID, RCODE)
      LATID = NCDID(NCID, 'lat', RCODE)
      CALL NCDREN(NCID, LATID, 'latitude', RCODE)
* leave define mode
      CALL NCENDF(NCID, RCODE)
```

7 Variables

Variables for a netCDF file are defined when the file is created, while the netCDF file 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 is referred to by a small integer called a variable ID. Attributes may be associated with a variable to specify such properties as units, fill values, maximum and minimum valid values, scaling factors, and offsets.

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 a hyperslab of values into a variable, given variable ID, corner indices, edge lengths, and a block of values.
- Put a generalized hyperslab of values into a variable, given variable ID, corner indices, edge lengths, stride vector, index mapping vector, and a block of values.
- Put values into record variables, given record number and pointers to blocks of values.
- Get a data value from a variable, given variable ID and indices.
- Get a hyperslab of values from a variable, given variable ID, corner indices, and edge lengths.
- Get a generalized hyperslab of values from a variable, given variable ID, corner indices, edge lengths, stride vector, and index mapping vector.
- Get values from record variables, given record number and pointers to where the data should be stored for each record variable.
- Rename a variable.
- Get number of bytes for a given data type.
- Get the number of record variables, their IDs, and their record sizes.

7.1 Create a Variable

The function `ncvardef` (or `NCVDEF` for FORTRAN) adds a new variable to an open netCDF file in define mode. It returns a variable ID, given the netCDF ID, the variable name, the variable type, the number of dimensions, and a list of the dimension IDs.

In case of an error, `ncvardef` returns -1; `NCVDEF` returns a nonzero value in `rcode`. Possible causes of errors include:

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

ncvardef: C Interface

```
int ncvardef(int ncid, const char* name, nc_type datatype,
             int ndims, const int dimids[]);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`name` Variable name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant.

`datatype` One of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`.

`ndims` 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 `MAX_VAR_DIMS`.

`dimids` Vector of `ndims` dimension IDs corresponding to the variable dimensions. If the ID of the unlimited dimension is included, it must be first.

Here is an example using `ncvardef` to create a variable named `rh` of type `long` with three dimensions, `time`, `lat`, and `lon` in a new netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;                          /* netCDF ID */
int  lat_dim, lon_dim, time_dim;    /* dimension IDs */
int  rh_id;                         /* variable ID */
int  rh_dimids[3];                  /* variable shape */
   ...
ncid = nccreate("foo.nc", NC_CLOBBER);
   ...
                                    /* define dimensions */
lat_dim = ncdimdef(ncid, "lat", 5L);
lon_dim = ncdimdef(ncid, "lon", 10L);
time_dim = ncdimdef(ncid, "time", NC_UNLIMITED);
   ...
                                   /* define variable */
rh_dimids[0] = time_dim;
rh_dimids[1] = lat_dim;
rh_dimids[2] = lon_dim;
rh_id = ncvardef (ncid, "rh", NC_DOUBLE, 3, rh_dimids);
```

NCVDEF: FORTRAN Interface

```
      INTEGER FUNCTION NCVDEF(INTEGER NCID, CHARACTER*(*) VARNAM,
     +                        INTEGER VARTYP, INTEGER NVDIMS,
     +                        INTEGER VDIMS(*), INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARNAM` Variable name. Must begin with an alphabetic character, which is followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant.

`VARTYP` One of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`.

`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 `MAXVDIMS`.

`VDIMS` Vector of `NVDIMS` dimension IDs corresponding to the variable dimensions. If the ID of the unlimited dimension is included, it must be last.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVDEF` to create a variable named `rh` of type `long` with three dimensions, `time`, `lat`, and `lon` in a new netCDF file named '`foo.nc`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  LATDIM, LONDIM, TIMDIM ! dimension IDs
INTEGER  RHID                   ! variable ID
INTEGER  RHDIMS(3)              ! variable shape
   ...
NCID = NCCRE ('foo.nc', NC_CLOBBER, RCODE)
   ...
                                ! define dimensions
LATDIM = NCDDEF(NCID, 'lat', 5, RCODE)
LONDIM = NCDDEF(NCID, 'lon', 10, RCODE)
TIMDIM = NCDDEF(NCID, 'time', NCUNLIM, RCODE)
   ...
                                ! define variable
RHDIMS(1) = LONDIM
RHDIMS(2) = LATDIM
RHDIMS(3) = TIMDIM
RHID = NCVDEF (NCID, 'rh', NCDOUBLE, 3, RHDIMS, RCODE)
```

7.2 Get a Variable ID from Its Name

The function `ncvarid` (or `NCVID` for FORTRAN) returns the ID of a netCDF variable, given its name.

In case of an error, `ncvarid` returns -1; `NCVID` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified variable name is not a valid name for a variable in the specified netCDF file.
- The specified netCDF ID does not refer to an open netCDF file.

ncvarid: C Interface

```
int ncvarid(int ncid, const char* name);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`name` Variable name for which ID is desired.

Here is an example using `ncvarid` to find out the ID of a variable named `rh` in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;                          /* netCDF ID */
int  rh_id;                         /* variable ID */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
```

NCVID: FORTRAN Interface

```
      INTEGER FUNCTION NCVID(INTEGER NCID,
     +                       CHARACTER*(*) VARNAM,
```

```
+                           INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARNAM` Variable name for which ID is desired.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVID` to find out the ID of a variable named `rh` in an existing netCDF file named '`foo.nc`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  RHID                    ! variable ID
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)
```

7.3 Get Information about a Variable from Its ID

The function `ncvarinq` (or `NCVINQ` for FORTRAN) returns information about a netCDF variable, given its ID. The information returned is the 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.

In case of an error, `ncvarinq` returns -1; `NCVINQ` returns a nonzero value in `rcode`. Possible causes of errors include:

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

ncvarinq: C Interface

```
int ncvarinq(int ncid, int varid, char* name, nc_type* datatype,
             int* ndims, int dimids[], int* natts);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID, returned from a previous call to `ncvardef` or `ncvarid`.

`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 `MAX_NC_NAME`. If the name parameter is given as '`(char *) 0`', no name will be returned so no space needs to be allocated.

`datatype` Returned variable type, one of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`. If this parameter is given as '`(nc_type *) 0`', no type will be returned so no variable to hold the type needs to be declared.

`ndims` Returned number of dimensions the variable was defined as using. For example, `2` specifies a matrix, `1` specifies a vector, and `0` means the variable is a scalar with no dimensions. If this parameter is given as '`(int *) 0`', no number of dimensions will be returned so no variable to hold this information needs to be declared.

`dimids` Returned vector of `ndims` dimension IDs corresponding to the variable dimensions. The caller must allocate enough space for a vector of at least `ndims` integers to be returned.

The maximum possible number of dimensions for a variable is given by the predefined constant `MAX_VAR_DIMS`. If this parameter is given as '`(int *) 0`', no vector will be returned so no space to hold the dimension IDs needs to be declared or allocated.

`natts` Returned number of variable attributes assigned to this variable. If this parameter is given as '`(int *) 0`', the number of attributes will not be returned so no space to hold this information needs to be declared or allocated.

Here is an example using `ncvarinq` to find out about a variable named `rh` in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;                         /* netCDF ID */
int  rh_id;                        /* variable ID */
nc_type rh_type;                   /* variable type */
int rh_ndims;                      /* number of dims */
int  rh_dims[MAX_VAR_DIMS];        /* variable shape */
int rh_natts                       /* number of attributes */
   ...
ncid = ncopen ("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
/* we don't need name, since we already know it */
ncvarinq (ncid, rh_id, (char *) 0, &rh_type, &rh_ndims, rh_dims,
          &rh_natts);
```

NCVINQ: FORTRAN Interface

```
      SUBROUTINE NCVINQ (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) VARNAM, INTEGER VARTYP,
     +                   INTEGER NVDIMS, INTEGER VDIMS(*),
     +                   INTEGER NVATTS, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID, returned from a previous call to `NCVDEF` or `NCVID`.

`VARNAM` 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 `MAXNCNAM`.

`VARTYP` Returned variable type, one of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`.

`NVDIMS` Returned 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.

`VDIMS` Returned vector of `NVDIMS` dimension IDs corresponding to the variable dimensions. The caller must allocate enough space for a vector of at least `NVDIMS` integers to be returned. The maximum possible number of dimensions for a variable is given by the predefined constant `MAXVDIMS`.

`NVATTS` Returned number of variable attributes assigned to this variable.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVINQ` to find out about a variable named `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID, RCODE
      INTEGER  RHID                 ! variable ID
      CHARACTER*31 RHNAME           ! variable name
      INTEGER  RHTYPE               ! variable type
      INTEGER  RHN                  ! number of dimensions
      INTEGER  RHDIMS(MAXVDIMS)     ! variable shape
      INTEGER  RHNATT               ! number of attributes
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)! get ID
      CALL NCVINQ (NCID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT,
     +             RCODE)
```

7.4 Write a Single Data Value

The function `ncvarput1` (or `NCVPT1` or `NCVP1C` for FORTRAN) puts a single data value into a variable of an open netCDF file that is in data mode. Inputs are the netCDF ID, the variable ID, a multidimensional index that specifies which value to add or alter, and the data value.

In case of an error, `ncvarput1` returns -1; `NCVPT1` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified indices were out of range for the dimensionality of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension size will cause an error.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

ncvarput1: C Interface

```
int ncvarput1(int ncid, int varid, const long mindex[], const void *value);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID, returned from a previous call to `ncvardef` or `ncvarid`.

`mindex` The multidimensional index of the the data value to be written. The indices are relative to 0, so for example, the first data value of a two-dimensional variable would have index `(0,0)`. The elements of `mindex` must correspond to the variable's dimensions. Hence, if the variable is a record variable, the first index would correspond to the record number.

`value` Pointer to the data value to be written. The pointer is declared to be of type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvarput1` to set the `(1,2,3)` element of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `time`, `lat`, and `lon`, so we want to set the value of `rh` that corresponds to the second `time` value, the third `lat` value, and the fourth `lon` value:

```
#include "netcdf.h"
   ...
int  ncid;                          /* netCDF ID */
int  rh_id;                         /* variable ID */
static long rh_index[] = {1, 2, 3}; /* where to put value */
static double rh_val = 0.5;         /* value to put */
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
ncvarput1(ncid, rh_id, rh_index, (void *) &rh_val);
```

NCVPT1: FORTRAN Interface

```
      SUBROUTINE NCVPT1 (INTEGER NCID, INTEGER VARID,
     +                   INTEGER MINDEX(*), type VALUE,
     +                   INTEGER RCODE)

      SUBROUTINE NCVP1C (INTEGER NCID, INTEGER VARID,
     +                   INTEGER MINDEX(*), CHARACTER CHVAL,
     +                   INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVPT1` and `NCVP1C`, for putting a single value in a variable. The first puts a numeric value in a variable of numeric type, and the second puts a character value in a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID, returned from a previous call to `NCVDEF` or `NCVID`.

`MINDEX` The multidimensional index of the 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 `mindex` must correspond to the variable's dimensions. Hence, if the variable is a record variable, the last index would correspond to the record number.

`VALUE` For `NCVPT1`, the data value to be written. The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`CHVAL` For `NCVP1C`, the data value to be written. The data should be of a type character, corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVPT1` to set the `(4,3,2)` element of the variable named `rh` to `0.5` in an existing netCDF file 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  NCID, RCODE
INTEGER  RHID                ! variable ID
INTEGER  RHINDX(3)           ! where to put value
DATA RHINDX /4, 3, 2/
   ...
NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)  ! get ID
CALL NCVPT1 (NCID, RHID, RHINDX, 0.5, RCODE)
```

7.5 Write a Hyperslab of Values

The function `ncvarput` (or `NCVPT` or `NCVPTC` for FORTRAN) writes a hyperslab of values into a netCDF variable of an open netCDF file. The hyperslab is specified by giving a corner and a vector of edge lengths. The values are specified as a vector whose elements are ordered by assuming that the last dimension of the hyperslab varies fastest for C, the first dimension varies fastest for FORTRAN. The netCDF file must be in data mode.

In case of an error, `ncvarput` returns -1; `NCVPT` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified corner indices were out of range for the dimensionality of the specified variable. For example, a negative index, or an index that is larger than the corresponding dimension size will cause an error.
- The specified edge lengths added to the specified corner would have referenced data out of range for the dimensionality of the specified variable. For example, an edge length that is larger than the corresponding dimension size minus the corner index will cause an error.
- The specified netCDF file is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

ncvarput: C Interface

```
int ncvarput(int ncid, int varid, const long start[], const long count[],
             const void *values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID, returned from a previous call to `ncvardef` or `ncvarid`.

`start` A vector of long integers specifying the multidimensional index of the corner of the hyperslab where the first of the data values will be written. The indices are relative to 0, so for example, the first data value of a variable would have index `(0, 0, ..., 0)`. The size 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 first index would correspond to the starting record number for writing the data values.

`count` A vector of long integers specifying the multidimensional edge lengths from the corner of the hyperslab where the first of the data values will be written. To write a single value, for example, specify `count` as `(1, 1, ..., 1)`. The size 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 first element of `count` corresponds to a count of the number of records to write.

`value` Pointer to a block of data values to be written. The order in which the data will be written to the netCDF variable is with the last dimension of the specified hyperslab varying fastest. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvarput` to add or change all the values of the variable named `rh` to `0.5` in an existing netCDF file 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.h"
   ...
#define TIMES 3
#define LATS  5
#define LONS  10
int  ncid;                         /* netCDF ID */
int  rh_id;                        /* variable ID */
static long start[] = {0, 0, 0};    /* start at first value */
static long count[] = {TIMES, LATS, LONS};
double rh_vals[TIMES*LATS*LONS];   /* array to hold values */
int i;
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
for (i = 0; i < TIMES*LATS*LONS; i++)
    rh_vals[i] = 0.5;
/* write hyperslab of values into netCDF variable */
ncvarput(ncid, rh_id, start, count, (void *) rh_vals);
```

NCVPT: FORTRAN Interface

```
 SUBROUTINE NCVPT (INTEGER NCID, INTEGER VARID,
+                  INTEGER START(*), INTEGER COUNT(*),
+                  type VALUES, INTEGER RCODE)

 SUBROUTINE NCVPTC(INTEGER NCID, INTEGER VARID,
+                  INTEGER START(*), INTEGER COUNTS(*),
+                  CHARACTER*(*) STRING, INTEGER LENSTR,
+                  INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVPT` and `NCVPTC`, for writing a hyperslab of values into a netCDF variable. The first writes numeric values into a variable of numeric type, and the second writes character values into a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID, returned from a previous call to `NCVDEF` or `NCVID`.

`START` A vector of integers specifying the multidimensional index of the corner of the hyperslab 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 size 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 multidimensional edge lengths from the corner of the hyperslab where the first of the data values will be written. To write a single value, for example, specify `COUNT` as `(1, 1, ..., 1)`. The size 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.

`VALUES` For `NCVPT`, the block of data values to be written. The order in which the data will be written into the specified hyperslab is with the first dimension varying fastest (like the ordinary FORTRAN convention). The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`STRING` For `NCVPTC`, the characters to be written. The order in which the characters will be written to the netCDF variable is with the first dimension of the specified hyperslab varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

`LENSTR` For `NCVPTC`, the total declared length (in characters) of the `STRING` argument. This should be at least as large as the product of the elements of the `COUNT` vector. Note that this is not necessarily the same as the value returned by the FORTRAN `LEN` function, because an array argument may be provided.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVPT` to add or change all the values of the variable named `rh` to `0.5` in an existing netCDF file 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 sizes
      INTEGER  NCID, RCODE
      INTEGER  RHID               ! variable ID
      INTEGER  START(NDIMS), COUNT(NDIMS) ! hyperslab
      DOUBLE RHVALS(LONS, LATS, TIMES)
      DATA START /1, 1, 1/        ! start at first value
      DATA COUNT /LONS, LATS, TIMES/
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)   ! get ID
      DO 10 ILON = 1, LONS
         DO 10 ILAT = 1, LATS
            DO 10 ITIME = 1, TIMES
               RHVALS(ILON, ILAT, ITIME) = 0.5
   10 CONTINUE
      CALL NCVPT (NCID, RHID, START, COUNT, RHVALS, RCODE)
```

7.6 Write a Generalized Hyperslab of Values

The function `ncvarputg` (or `NCVPTG` or `NCVPGC` for FORTRAN) writes a generalized hyperslab of values into a netCDF variable of an open netCDF file. The generalized hyperslab is specified by giving a corner, a vector of edge lengths, a stride vector, and an index mapping vector. No assumptions are made about the ordering or size of the dimensions of the data array. The netCDF file must be in data mode.

In case of an error, `ncvarputg` returns -1; `NCVPTG` and `NCVPGC` return a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified corner indices were out of range for the dimensionality of the specified variable. For example, a negative index, or an index that is larger than the corresponding dimension size will cause an error.
- The specified edge lengths and strides added to the specified corner would have referenced data out of range for the dimensionality of the specified variable. For example, an edge length that is larger than the corresponding dimension size minus the corner index will cause an error, as will accessing two or more points using a stride that is greater than the size of the netCDF variable in the corresponding dimension.
- A non-positive stride.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

ncvarputg: C Interface

```
int ncvarputg(int ncid, int varid, const long start[], const long count[],
              const long stride[], const long imap[], const void *values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID, returned from a previous call to `ncvardef` or `ncvarid`.

`start` A vector of long integers specifying the multidimensional index of the corner of the hyperslab where the first of the data values will be written. The indices are relative to 0, so for example, the first data value of a variable would have index `(0, 0, ..., 0)`. The size 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 first index would correspond to the starting record number for writing the data values.

`count` A vector of long integers specifying the multidimensional edge lengths from the corner of the hyperslab where the first of the data values will be written. To write a single value, for example, specify `count` as `(1, 1, ..., 1)`. The size 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 first element of `count` corresponds to a count of the number of records to write.

`stride` A vector of long integers specifying, for each dimension, the interval between the accessed values of a netCDF variable. The size of the vector shall be at least the number of dimensions of the associated netCDF variable and its elements shall 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 `NULL` stride argument obtains the default behavior in which adjacent values are accessed along each dimension.

`imap` A vector of long integers specifying, for each dimension, how data values associated with a netCDF variable are arranged in memory. The offset, in bytes, from the memory location pointed to by the `value` argument to a particular datum is given by the inner product of the index mapping vector with the coordinates of the datum. (The *inner product* of two vectors [x0, x1, ..., xn] and [y0, y1, ..., yn] is just x0*y0 + x1*y1 + ... + xn*yn.) The vector may contain negative values if the `value` argument is appropriately specified. A `NULL` argument obtains the default behavior in which the memory-resident values are assumed to have the same structure as the associated netCDF variable.

`value` Pointer to a block of data values to be written. The order in which the data will be written to the netCDF variable is with the last dimension of the generalized hyperslab varying fastest. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvarputg` to add or change every other value in each dimension of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. Values are taken, using the same dimensional strides, from points in a 3-dimensional array of structures whose dimensions are the reverse of the netCDF variable. 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.h"
   ...
#define TIMES 3
#define LATS  5
#define LONS  10
int  ncid;                         /* netCDF ID */
int  rh_id;                        /* variable ID */
static long start[] = {0, 0, 0};   /* start at first value */
static long count[] = {TIMES, LATS, LONS};
static long stride[] = {2, 2, 2};  /* every other value */
long imap[3];                      /* set to reverse of variable */
struct datum {
    int    dummy;                  /* to illustrate mapping vector */
    double rh_val;                 /* actual value to be written */
}      data[LONS][LATS][TIMES];    /* reversed array to hold values. */
int itime, ilat, ilon;
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
for (ilon = 0; ilon < LONS; ilon += stride[2])
    for (ilat = 0; ilat < LATS; ilat += stride[1])
        for (itime = 0; itime < TIMES; itime += stride[0])
            data[ilon][ilat][itime].rh_val = 0.5;
/* access every ‘stride’ in-memory value using reversed dimensions */
imap[0] = stride[2]*sizeof(struct datum);
imap[1] = stride[1]*(1+(LONS-1)/stride[0])*imap[0];
imap[2] = stride[0]*(1+(LATS-1)/stride[1])*imap[1];
/* write generalized hyperslab of values into netCDF variable */
ncvarputg(ncid, rh_id, start, count, stride, imap, (void*)&data[0][0][0].rh_val);
```

NCVPTG, NCVPGC: FORTRAN Interface

```
      SUBROUTINE NCVPTG (INTEGER NCID, INTEGER VARID,
     +                   INTEGER START(*), INTEGER COUNT(*),
     +                   INTEGER STRIDE(*), INTEGER IMAP(*),
     +                   type VALUES, INTEGER RCODE)

      SUBROUTINE NCVPGC (INTEGER NCID, INTEGER VARID,
     +                   INTEGER START(*), INTEGER COUNT(*),
     +                   INTEGER STRIDE(*), INTEGER IMAP(*),
     +                   CHARACTER*(*) STRING, INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVPTG` and `NCVPGC`, for writing a generalized hyperslab of values into a netCDF variable. The first writes numeric values into a variable of numeric type, and the second writes character values into a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID, returned from a previous call to `NCVDEF` or `NCVID`.

`START` A vector of integers specifying the multidimensional index of the corner of the hyperslab 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 size 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 multidimensional edge lengths from the corner of the hyperslab where the first of the data values will be written. To write a single value, for example, specify `COUNT` as `(1, 1, ..., 1)`. The size 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.

`STRIDE` A vector of integers specifying, for each dimension, the interval between the accessed values of a netCDF variable or the value `0`. The size of the vector shall be at least the number of dimensions of the associated netCDF variable and its elements shall 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. An `0` argument obtains the default behavior in which adjacent values are accessed along each dimension.

`IMAP` A vector of long integers specifying, for each dimension, how data values associated with a netCDF variable are arranged in memory or the value `0`. The offset, in bytes, from the memory location pointed to by the `value` argument to a particular datum is given by the inner product of the index mapping vector with the (origin-0) coordinates of the datum. (The *inner product* of two vectors [x1, x2, ..., xn] and [y1, y2, ..., yn] is just x1*y1 + x2*y2 + ... + xn*yn.) The vector may contain negative values if the `value` argument is appropriately specified. A `0` argument obtains the default behavior in which the memory-resident values are assumed to have the same structure as the associated netCDF variable.

`VALUES` For `NCVPTG`, the block of data values to be written. The order in which the data will be written from the specified hyperslab is with the first dimension of the generalized hyperslab varying fastest (like the ordinary FORTRAN convention). The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`STRING` For `NCVPGC`, the characters to be written. The order in which the characters will be written to the netCDF variable is with the first dimension of the generalized hyperslab varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVPTG` to add or change every other value in each dimension of the variable named `rh` to `0.5` in an existing netCDF file named '`foo.nc`'. Values are taken, using the same dimensional strides, from a 2-parameter array whose dimensions are the reverse of the netCDF variable. 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 sizes
      INTEGER  NCID, RCODE
      INTEGER  RHID               ! variable ID
      INTEGER  START(NDIMS), COUNT(NDIMS),
     +         STRIDE(NDIMS), IMAP(NDIMS)  ! generalized hyperslab
      DOUBLE DATA(2, TIMES, LATS, LONS)    ! rh is second parameter
      DATA START /1, 1, 1/        ! start at first value
      DATA COUNT /LONS, LATS, TIMES/
      DATA STRIDE /2, 2, 2/
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)   ! get ID
      DO 10 ILON = 1, LONGS, STRIDE(1)
         DO 10 ILAT = 1, LATS, STRIDE(2)
            DO 10 ITIME = 1, TIMES, STRIDE(3)
               DATA(2, ITIME, ILAT, ILON) = 0.5
   10 CONTINUE
      IMAP(3) = 8*2*2   ! every other point of vector of 2-doubles
      IMAP(2) = IMAP(3)*(1+(TIMES-1)/STRIDE(3))*2
      IMAP(1) = IMAP(2)*(1+(LATS-1)/STRIDE(2))*2
      CALL NCVPTG (NCID, RHID, START, COUNT, STRIDE, IMAP,
     +             DATA(2,1,1,1), RCODE)
```

7.7 Put a Record

The function `ncrecput` writes a multi-variable record of values (or part of a record of values) into the record variables of an open netCDF file. The record is specified by giving a record number. The values to be written are specified by an array of pointers, one for each record variable, to blocks of values. Each block of values should be of the appropriate size and type for a record's worth of data for the corresponding record variable. Each such pointer must be either NULL, to indicate that no data is to be written for that variable, or must point to an entire record's worth of data of the appropriate type for the corresponding record variable. The values for each record variable are assumed to be ordered with the last dimension varying fastest. The netCDF file must be in data mode.

The `ncrecput` function is not strictly necessary, since the same data may be written with a sequence of calls to `ncvarput`, one for each record variable for which a non-NULL pointer is specified. This function is provided in the C interface for convenience only; no corresponding Fortran interface is available, so Fortran users should use multiple calls to `NCVPT` or `NCVPTC` instead.

To use `ncrecput` properly, you must know the number, order, and types of record variables in the netCDF file, information that can be determined with a call to `ncrecinq`. If your assumptions about the number, order, or types of record variables in the file is incorrect, calling this function may lead to incorrect results or even a segmentation violation. **Warning: neither the compiler nor the netCDF software can detect errors with the pointer array argument to `ncrecput`.**

In case of a detected error, `ncrecput` returns -1. Possible causes of detectable errors include:

- The specified record number is less than zero.
- The specified netCDF file is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

ncrecput: C Interface

```
int ncrecput(int ncid, long recnum, const void *datap[]);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`recnum` Record number, specifying the value of the unlimited dimension for which data is to be written. The first record is record number `0`. Note that if you specify a value for recnum that is larger than the current size of the unlimited dimension, intervening records will be written with fill values before the data is written in the specified record, unless ncsetfill has been called to specify no prefilling.

datap
: Array of pointers to blocks of data values to be written, one for each record variable. datap[i], if non-NULL, must point to an entire record's worth of data for the i-th record variable. For NULL pointers, no data will be written for the corresponding record variables. This permits you to specify an arbitrary subset of record variables. The data pointed to should be of the appropriate type for each record variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using ncrecput to write the value of a C struct into a netCDF file with a single call. This example assumes that record variables of the appropriate shapes and types have previously been created in the netCDF file.

```
#include "netcdf.h"
   ...
    static struct {
    char city[20];
    long date;
    float lat;
    float lon;
    float precip[24];            /* hourly precipitation */
    } rec = {
        "Pocatello",
        930228,
        42.92,
        -112.60,
        {0,0,.1,.2,.2,.3,.2,0,0,0,0,0,0,0,0,0,.3,1.1,0,0,0,0,0,0}
    };

    int ncid;                    /* id of open netcdf file */
    long recnum;                 /* number of record to write */
    void *datap[5];              /* array of address pointers for record
vars */
    ...
    datap[0] = &rec.city[0];
    datap[1] = &rec.date;
    datap[2] = &rec.lat;
    datap[3] = &rec.lon;
    datap[4] = &rec.precip[0];

    ncrecput(ncid, recnum, datap);  /* instead of 5 calls to ncvarget */
```

7.8 Read a Single Data Value

The function `ncvarget1` (or `NCVGT1` or `NCVG1C` for FORTRAN) gets a single data value from a variable of an open netCDF file 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.

In case of an error, `ncvarget1` returns -1; `NCVGT1` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified indices were out of range for the dimensionality of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension size will cause an error.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

ncvarget1: C Interface

```
int ncvarget1(int ncid, int varid, const long mindex[], void *value);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID, returned from a previous call to `ncvardef` or `ncvarid`.

`mindex` The multidimensional index of the the data value to be read. The indices are relative to 0, so for example, the first data value of a two-dimensional variable would have index `(0,0)`. The elements of `mindex` must correspond to the variable's dimensions. Hence, if the variable is a record variable, the first index is the record number.

`value` Pointer to the location into which the data value is read. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type for the data value is used.**

Here is an example using `ncvarget1` to get the `(1,2,3)` element of the variable named `rh` in an existing netCDF file named '`foo.nc`'. For simplicity in this example, we assume that we know that `rh` is dimensioned with `time`, `lat`, and `lon`, so we want to get the value of `rh` that corresponds to the second `time` value, the third `lat` value, and the fourth `lon` value:

```
#include "netcdf.h"
   ...
int ncid;                              /* netCDF ID */
int rh_id;                             /* variable ID */
static long rh_index[] = {1, 2, 3};  /* where to get value from */
double rh_val;                         /* where to put it */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
ncvarget1(ncid, rh_id, rh_index, (void *) &rh_val);
```

NCVGT1: FORTRAN Interface

```
 SUBROUTINE NCVGT1 (INTEGER NCID, INTEGER VARID,
+ INTEGER MINDEX(*), type VALUE,
+ INTEGER RCODE)

 SUBROUTINE NCVG1C (INTEGER NCID, INTEGER VARID,
+ INTEGER MINDEX(*), CHARACTER CHVAL,
+ INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVGT1` and `NCVG1C`, for reading a single value from a variable. The first reads a numeric value in a variable of numeric type, and the second reads a character value in a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID, returned from a previous call to `NCVDEF` or `NCVID`.

`MINDEX` The multidimensional index of the 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 `mindex` correspond to the variable's dimensions. Hence, if the variable is a record variable, the last index is the record number.

`VALUE` For `NCVGT1`, the location into which the data value will be read. The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`CHVAL` For `NCVG1C`, the location into which the data value will be read. This should be of a type character, corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVGT1` to get the `(4,3,2)` element of the variable named `rh` in an existing netCDF file 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 NCID, RCODE
INTEGER RHID ! variable ID
INTEGER RHINDX(3) ! where to get value
DOUBLE PRECISION RHVAL ! put it here
DATA RHINDX /4, 3, 2/
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)! get ID
CALL NCVGT1 (NCID, RHID, RHINDX, RHVAL, RCODE)
```

7.9 Read a Hyperslab of Values

The function `ncvarget` (or `NCVGT` or `NCVGTC` for FORTRAN) reads a hyperslab of values from a netCDF variable of an open netCDF file. The hyperslab is specified by giving a corner and a vector of edge lengths. The values are read into consecutive locations with the last (or first for FORTRAN) dimension of the hyperslab varying fastest. The netCDF file must be in data mode.

In case of an error, `ncvarget` returns -1; `NCVGT` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified corner indices were out of range for the dimensionality of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension size will cause an error.
- The specified edge lengths added to the specified corner would have referenced data out of range for the dimensionality of the specified variable. For example, an edge length that is larger than the corresponding dimension size minus the corner index will cause an error.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

ncvarget: C Interface

```
int ncvarget(int ncid, int varid, const long start[], const long count[],
             void *values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID, returned from a previous call to `ncvardef` or `ncvarid`.

`start` A vector of long integers specifying the multidimensional index of the corner of the hyperslab where the first of the data values will be read from. The indices are relative to 0, so for example, the first data value of a variable would have index `(0, 0, ..., 0)`. The size 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 first index would correspond to the starting record number for reading the data values.

`count` A vector of long integers specifying the multidimensional edge lengths from the corner of the hyperslab where the first of the data values will be read. To read a single value, for example, specify `count` as `(1, 1, ..., 1)`. The size 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 first element of `count` corresponds to a count of the number of records to read.

`value` Pointer to the first of the locations into which the data values will be read. The order in which the data will be read from the netCDF variable is with the last dimension of the specified hyperslab varying fastest. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvarget` to read all the values of the variable named `rh` from an existing netCDF file 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.h"
   ... #define TIMES 3 #define LATS 5 #define LONS 10 int ncid; /*
netCDF ID */ int rh_id; /* variable ID */ static long start[] = {0, 0,
0}; /* start at first value */ static long count[] = {TIMES, LATS,
LONS}; double rh_vals[TIMES*LATS*LONS]; /* array to hold values */
   ... ncid = ncopen("foo.nc", NC_NOWRITE);
   ... rh_id = ncvarid (ncid, "rh");
   ... /* read hyperslab of values from netCDF variable */
ncvarget(ncid, rh_id, start, count, (void *) rh_vals);
```

NCVGT, NCVGTC: FORTRAN Interface

```
 SUBROUTINE NCVGT (INTEGER NCID, INTEGER VARID,
+ INTEGER START(*), INTEGER COUNT(*),
+ type VALUES, INTEGER RCODE)

 SUBROUTINE NCVGTC(INTEGER NCID, INTEGER VARID,
+ INTEGER START(*), INTEGER COUNTS(*),
+ CHARACTER*(*) STRING, INTEGER LENSTR,
+ INTEGER RCODE)
```

There are two FORTRAN subroutines, NCVGT and NCVGTC, for reading a hyperslab of values from a netCDF variable. The first reads numeric values from a variable of numeric type, and the second reads character values from a variable of character type.

NCID
: NetCDF ID, returned from a previous call to NCOPN or NCCRE.

VARID
: Variable ID, returned from a previous call to NCVDEF or NCVID.

START
: A vector of integers specifying the multidimensional index of the corner of the hyperslab 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 size 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 reading the data values.

COUNT
: A vector of integers specifying the multidimensional edge lengths from the corner of the hyperslab where the first of the data values will be read. To read a single value, for example, specify COUNT as (1, 1, ..., 1). The size 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 read.

VALUES
: For NCVGT, the locations into which the data values will be read. The order in which the data will be read from the netCDF variable is with the first dimension of the specified hyperslab varying fastest (like the ordinary FORTRAN convention). The data may be of a type corresponding to any of the netCDF types NCSHORT, NCLONG, NCFLOAT, or NCDOUBLE, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

STRING
: For NCVGTC, the character string into which the character data will be read. The order in which the characters will be read from the netCDF variable is with the first dimension of the specified hyperslab varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types NCCHAR or NCBYTE.

LENSTR
: For NCVGTC, the total declared length (in characters) of the STRING argument. This should be at least as large as the product of the elements of the COUNT vector. Note that this is not necessarily the same as the value returned by the FORTRAN LEN function, because an array argument may be provided. NCVGTC will check to make sure the requested data will fit in LENSTR characters.

RCODE
: Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVGT` to read all the values of the variable named `rh` from an existing netCDF file 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 sizes
INTEGER NCID, RCODE
INTEGER RHID ! variable ID
INTEGER START(NDIMS), COUNT(NDIMS) ! hyperslab
DOUBLE RHVALS(LONS, LATS, TIMES)
DATA START /1, 1, 1/ ! start at first value
DATA COUNT /LONS, LATS, TIMES/
   ...
NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
   ...
RHID = NCVID (NCID, 'rh', RCODE)! get ID
CALL NCVGT (NCID, RHID, START, COUNT, RHVALS, RCODE)
```

7.10 Read a Generalized Hyperslab of Values

The function `ncvargetg` (or `NCVGTG` or `NCVGGC` for FORTRAN) reads a generalized hyperslab of values from a netCDF variable of an open netCDF file. The generalized hyperslab is specified by giving a corner, a vector of edge lengths, a stride vector, and an index mapping vector. The values are read with the last (or first for FORTRAN) dimension of the generalized hyperslab varying fastest. The netCDF file must be in data mode.

In case of an error, `ncvargetg` returns -1; `NCVGTG` and `NCVGGC` return a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified corner indices were out of range for the dimensionality of the specified variable. For example, a negative index or an index that is larger than the corresponding dimension size will cause an error.
- The specified edge lengths and strides added to the specified corner would have referenced data out of range for the dimensionality of the specified variable. For example, an edge length that is larger than the corresponding dimension size minus the corner index will cause an error, as will accessing two or more points using a stride that is greater than the size of the netCDF variable in the corresponding dimension.
- A non-positive stride.
- The specified netCDF is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

ncvargetg: C Interface

```
int ncvargetg(int ncid, int varid, const long start[], const long count[],
              const long stride[], const long imap[], void *values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID, returned from a previous call to `ncvardef` or `ncvarid`.

`start` A vector of long integers specifying the multidimensional index of the corner of the hyperslab where the first of the data values will be read from. The indices are relative to 0, so for example, the first data value of a variable would have index `(0, 0, ..., 0)`. The size 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 first index would correspond to the starting record number for reading the data values.

`count` A vector of long integers specifying the multidimensional edge lengths from the corner of the hyperslab where the first of the data values will be read. To read a single value, for example, specify `count` as `(1, 1, ..., 1)`. The size 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 first element of `count` corresponds to a count of the number of records to read.

`stride` A vector of long integers specifying, for each dimension, the interval between the accessed values of a netCDF variable. The size of the vector shall be at least the number of dimensions of the associated netCDF variable and its elements shall 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 `NULL` stride argument obtains the default behavior in which adjacent values are accessed along each dimension.

`imap` A vector of long integers specifying, for each dimension, how data values associated with a netCDF variable are arranged in memory. The offset, in bytes, from the memory location pointed to by the `value` argument to a particular datum is given by the inner product of the index mapping vector with the coordinates of the datum. (The *inner product* of two vectors [x0, x1, ..., xn] and [y0, y1, ..., yn] is just x0*y0 + x1*y1 + ... + xn*yn.) The vector may contain negative values if the `value` argument is appropriately specified. A `NULL` argument obtains the default behavior in which the memory-resident values are assumed to have the same structure as the associated netCDF variable.

`value` Pointer to the first of the locations into which the data values will be read. The order in which the data will be read from the netCDF variable is with the last dimension of the generalized hyperslab varying fastest. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncvargetg` to read every other value in each dimension of the variable named `rh` from an existing netCDF file named '`foo.nc`'. Values are assigned, using the same dimensional strides, to points in a 3-dimensional array of structures whose dimensions are the reverse of the netCDF variable. 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.h"
   ...
#define TIMES 3
#define LATS  5
#define LONS 10
int ncid;                          /* netCDF ID */
int rh_id;                         /* variable ID */
static long start[] = {0, 0, 0};  /* start at first value */
static long count[] = {TIMES, LATS, LONS};
static long stride[] = {2, 2, 2}; /* every other value */
long imap[3];                      /* set to reverse of variable */
struct datum {
    int dummy;                     /* to illustrate mapping vector usage */
    double rh_val;                 /* actual value to be read */
}      data[TIMES][LATS][LONS];    /* array to hold values */
   ...
   ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
   rh_id = ncvarid (ncid, "rh");
   ...
   /* access every 'stride' in-memory value using reversed dimensions */
   imap[0] = stride[2]*sizeof(struct datum);
   imap[1] = stride[1]*(1+(LONS-1)/stride[0])*imap[0];
   imap[2] = stride[0]*(1+(LATS-1)/stride[1])*imap[1];
   /* read generalized hyperslab of values from netCDF variable */
   ncvargetg(ncid, rh_id, start, count, stride, imap, (void*)&data[0][0][0].rh_val);
   ...
```

NCVGTG, NCVGGC: FORTRAN Interface

```
      SUBROUTINE NCVGTG (INTEGER NCID, INTEGER VARID,
     + INTEGER START(*), INTEGER COUNT(*),
     + INTEGER STRIDE(*), INTEGER IMAP(*),
     + type VALUES, INTEGER RCODE)

      SUBROUTINE NCVGGC (INTEGER NCID, INTEGER VARID,
     + INTEGER START(*), INTEGER COUNT(*),
     + INTEGER STRIDE(*), INTEGER IMAP(*),
     + CHARACTER*(*) STRING, INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCVGTG` and `NCVGGC`, for reading a generalized hyperslab of values from a netCDF variable. The first reads numeric values from a variable of numeric type, and the second reads character values from a variable of character type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID, returned from a previous call to `NCVDEF` or `NCVID`.

`START` A vector of integers specifying the multidimensional index of the corner of the hyperslab 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 size 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 reading the data values.

`COUNT` A vector of integers specifying the multidimensional edge lengths from the corner of the hyperslab where the first of the data values will be read. To read a single value, for example, specify `COUNT` as `(1, 1, ..., 1)`. The size 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 read.

`STRIDE` A vector of integers specifying, for each dimension, the interval between the accessed values of a netCDF variable or the value `0`. The size of the vector shall be at least the number of dimensions of the associated netCDF variable and its elements shall 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. An `0` argument obtains the default behavior in which adjacent values are accessed along each dimension.

`IMAP` A vector of long integers specifying, for each dimension, how data values associated with a netCDF variable are arranged in memory or the value `0`. The offset, in bytes, from the memory location pointed to by the `value` argument to a particular datum is given by the inner product of the index mapping vector with the (origin-0) coordinates of the datum. (The *inner product* of two vectors [x1, x2, ..., xn] and [y1, y2, ..., yn] is just x1*y1 + x2*y2 + ... + xn*yn.) The vector may contain negative values if the `value` argument is appropriately specified. A `0` argument obtains the default behavior in which the memory-resident values are assumed to have the same structure as the associated netCDF variable.

`VALUES` For `NCVGTG`, the locations into which the data values will be read. The order in which the data will be read from the netCDF variable is with the first dimension of the generalized hyperslab varying fastest (like the ordinary FORTRAN convention). The data may be of a type corresponding to any of the netCDF types `NCSHORT`, `NCLONG`, `NCFLOAT`, or `NCDOUBLE`, but must be appropriate for the type of the netCDF variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

`STRING` For `NCVGGC`, the character string into which the character data will be read. The order in which the characters will be read from the netCDF variable is with the first dimension of the generalized hyperslab varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCVGTG` to read every other value in each dimension of the variable named `rh` from an existing netCDF file named '`foo.nc`'. Values are assigned, using the same dimensional strides, to a 2-parameter array whose dimensions are the reverse of the netCDF variable. 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 sizes
 INTEGER NCID, RCODE
 INTEGER RHID ! variable ID
 INTEGER START(NDIMS), COUNT(NDIMS)
+ STRIDE(NDIMS), IMAP(NDIMS) ! generalized hyperslab
 DOUBLE DATA(2, TIMES, LATS, LONS) ! rh is second parameter
 DATA START /1, 1, 1/ ! start at first value
 DATA COUNT /LONS, LATS, TIMES/
 DATA STRIDE /2, 2, 2/
    ...
 NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
    ...
 RHID = NCVID (NCID, 'rh', RCODE)! get ID
 IMAP(3) = 8*2*2 ! every other point of vector of 2-doubles
 IMAP(2) = IMAP(3)*(1+(TIMES-1)/STRIDE(3))*2
 IMAP(1) = IMAP(2)*(1+(LATS-1)/STRIDE(2))*2
 CALL NCVGTG (NCID, RHID, START, COUNT, STRIDE, IMAP,
+ DATA(2,1,1,1), RCODE)
```

7.11 Get a Record

The function `ncrecget` reads a multi-variable record of values (or part of a record of values) from the record variables of an open netCDF file. The record is specified by giving a record number. The locations into which the data will be read are specified by an array of pointers, one for each record variable, to blocks of data. Each block of data should be of the appropriate size and type for a record's worth of data for the corresponding record variable. Each such pointer must be either NULL, to indicate that no data is to be written for that variable, or must point to space for an entire record's worth of data of the appropriate type for the corresponding record variable. The values for each record variable will be ordered with the last dimension varying fastest. The netCDF file must be in data mode.

The `ncrecget` function is not strictly necessary, since the same data may be read with a sequence of calls to `ncvarget`, one for each record variable for which a non-NULL pointer is specified. This function is provided in the C interface for convenience only; no corresponding Fortran interface is available, so Fortran users should use multiple calls to `NCVGT` or `NCVGTC` instead.

To use `ncrecget` properly, you must know the number, order, and types of record variables in the netCDF file, information that can be determined with a call to `ncrecinq`. If your assumptions about the number, order, or types of record variables in the file is incorrect, calling this function may lead to incorrect results or even a segmentation violation. **Warning: neither the compiler nor the netCDF software can detect errors with the pointer array argument to `ncrecget`.**

In case of a detected error, `ncrecget` returns -1. Possible causes of detectable errors include:

- The specified record number is less than zero.
- The specified netCDF file is in define mode rather than data mode.
- The specified netCDF ID does not refer to an open netCDF file.

ncrecget: C Interface

```
int ncrecget(int ncid, long recnum, void *datap[]);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`recnum` Record number, specifying the value of the unlimited dimension for which data is to be read. The first record is record number `0`.

`datap` Array of pointers to blocks of data into which the requested values will be read, one for each record variable. `datap[i]`, if non-NULL, must point to enough space to hold an entire record's worth of data for the `i`-th record variable. For NULL pointers, no data will be read for the corresponding record variables. This permits you to specify an arbitrary subset of record variables. The data pointed to should be of the appropriate type for each record variable. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncrecget` to read values into several C arrays and scalars with a single call. This example assumes that record variables of the appropriate shapes and types have previously been created in the netCDF file.

```
#include "netcdf.h"
   ...
    static struct {
    char city[20];
    long date;
    float lat;
    float lon;
    float precip[24];
    } rec[10];

    int ncid;           /* id of open netcdf file */
    long i;             /* number of record to read */
    void *datap[5];     /* array of address pointers for record vars */
    ...

    /* Get first 10 records of data */
    for(i=0; i<10; i++) {
        datap[0] = &rec[i].city[0];
        datap[1] = &rec[i].date;
        datap[2] = &rec[i].lat;
        datap[3] = &rec[i].lon;
        datap[4] = &rec[i].precip[0];
        ncrecget(ncid, i, datap);  /* instead of 5 calls to ncvarget */
    }
```

7.12 Reading and Writing Character String Values

Character strings are not a primitive netCDF 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 hyperslab access must be used to read and write character strings as variable data in netCDF files. Furthermore, variable-length strings are not supported by the netCDF interface except by convention; for example, you may treat a null (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; no hyperslab access is necessary (or possible) for attributes. 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 last dimension for the variable in C, the first in FORTRAN). The size 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-size 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 hyperslab access. This requires that you specify both a corner and a vector of edge lengths. The character-position dimension at the corner should be zero (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, i.e., `(1, 1, ..., 1, n)` or `(n, 1, 1, ..., 1)` in FORTRAN.

C Interface

In C, fixed-size strings may be written to a netCDF file without the terminating null byte, to save space. Variable-length strings should be written *with* a terminating null byte so that the intended length of the string can be determined when it is later read.

Here is an example that defines a record variable, `tx`, for character strings and stores a character-string value into the third record using `ncvarput`. In this example, we assume the string variable and data are to be added to an existing netCDF file named '`foo.nc`' that already has an unlimited record dimension `time`.

```
#include "netcdf.h"
   ...
int  ncid;            /* netCDF ID */
int  chid;            /* dimension ID for char positions */
int  timeid;          /* dimension ID for record dimension */
int  tx_id;           /* variable ID */
#define TDIMS 2       /* dimensionality of tx variable */
int tx_dims[TDIMS];   /* variable shape */
long tx_start[TDIMS];
long tx_count[TDIMS];
static char tx_val[] =
        "example string"; /* string to be put */
   ...
ncid = ncopen("foo.nc", NC_WRITE);
ncredef(ncid);       /* enter define mode */
   ...
/* define character-position dimension for strings of max length 40 */
chid = ncdimdef(ncid, "chid", 40L);
   ...
/* define a character-string variable */
tx_dims[0] = timeid;
tx_dims[1] = chid;    /* character-position dimension last */
tx_id = ncvardef (ncid, "tx", NC_CHAR, TDIMS, tx_dims);
   ...
ncendef(ncid);       /* leave define mode */
   ...
/* write tx_val into tx netCDF variable in record 3 */
tx_start[0] = 3;      /* record number to write */
tx_start[1] = 0;      /* start at beginning of variable */
tx_count[0] = 1;      /* only write one record */
tx_count[1] = strlen(tx_val) + 1;  /* number of chars to write */
ncvarput(ncid, tx_id, tx_start, tx_count, (void *) tx_val);
```

FORTRAN Interface

In FORTRAN, fixed-size strings may be written to a netCDF file without a terminating character, to save space. Variable-length strings should follow the C convention of writing strings with a terminating null 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 subroutines 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 `NCVPTC` and `NCVGTC`. The actual length of the string is specified as the value of the hyperslab 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 `NCVPTC`. In this example, we assume the string variable and data are to be added to an existing netCDF file named '`foo.nc`' that already has an unlimited record dimension `time`.

```
      INCLUDE 'netcdf.inc'
         ...
      PARAMETER (TDIMS=2)    ! number of TX dimensions
      PARAMETER (TXLEN = 15) ! length of example string
      INTEGER  NCID, RCODE
      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) ! hyperslab
      CHARACTER*40 TXVAL     ! max length 40
      DATA TXVAL /'example string'/
         ...
      TXVAL(TXLEN:TXLEN) = CHAR(0)   ! null terminate
         ...
      NCID = NCOPN('foo.nc', NCWRITE, RCODE)
      CALL NCREDF(NCID, RCODE) ! enter define mode
         ...
* define character-position dimension for strings of max length 40
      CHID = NCDDEF(NCID, "chid", 40, RCODE)
         ...
* define a character-string variable
      TXDIMS[1] = CHID   ! character-position dimension first
      TXDIMS[2] = TIMEID
      TXID = NCVDEF(NCID, "tx", NCCHAR, TDIMS, TXDIMS, RCODE)
         ...
      CALL NCENDF(NCID, RCODE) ! leave define mode
         ...
* write txval into tx netCDF variable in record 3
      TSTART[1] = 0      ! 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
      CALL NCVPTC (NCID, TXID, TSTART, TCOUNT, TXVAL, 40, RCODE)
```

7.13 Missing Values

What happens when you try to read a value that was never written in an open netCDF file? 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 file that is not open for reading, if the variable ID is invalid for the specified netCDF file, or if the specified hyperslab is not properly within the range defined by the dimension sizes of the specified variable. Otherwise, reading a value that was not written returns a special *fill value* used to fill in any missing values when a netCDF variable is first written.

You may also ignore fill values and use the entire range of a netCDF 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 `ncsetfill` before writing. This may provide a significant performance gain for netCDF writes.

There are several reasons for using a fill value instead of an error return for missing data. First, the interface for hyperslab access would necessarily be more complex and slower if information had to be returned about whether each value read had been written. Since data may have been written in a different order from that in which it is later read, it is possibile that only a few values in a block of retrieved values were never written. Second, it is usually preferable to delay the detection of missing values until there is a need for the values, since they may not be used in subsequent computations. Finally, the use of missing values is a common way to represent data points outside the boundaries of irregular regions of data enclosed by a hyperslab, making it possible to handle such data in a simpler way than would be possible with a more compact representation that represented the boundary explicitly.

The default fill values for each type are defined in the include file '`netcdf.h`' (or '`netcdf.inc`' for FORTRAN). It is usually better to use your own fill value instead, by defining the attribute `_FillValue` for a variable before writing it. A disadvantage of the default fill values for floating-point and double-precision types is that they may be defined differently for different platforms, and may be difficult to compare with other values, since they are defined to be right at the edge of the valid floating-point number ranges for each machine.

Fill values are used for filling in missing data whenever a value is put beyond the end of data that has already been written. A default fill value has no other special meaning, so it can be used for valid values if you use your own fill value instead.

Currently the only difference between the netCDF byte and character types is that the two types have different default fill values. The fill value for bytes is on the edge of the range, representing the largest negative value for signed bytes. The fill value for characters, however, is the zero byte, a more useful value for detecting the end of C character strings.

Sometimes there is need for more than one value to represent different kinds of missing data. In this case, the user should use one or more other variable attributes for the different kinds of missing data. For example, it might be appropriate to use `_FillValue` to mean that data that was expected never appeared, but `missing_value` where the creator of the data intends data to be missing, as around an irregular region represented by a rectangular grid.

7.14 Rename a Variable

The function `ncvarrename` (or `NCVREN` for FORTRAN) changes the name of a netCDF variable in an open netCDF. If the new name is longer than the old name, the netCDF must be in define mode. You cannot rename a variable to have the name of any existing variable.

In case of an error, `ncvarrename` returns -1; `NCVREN` returns a nonzero value in `rcode`. 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 file.
- The specified netCDF ID does not refer to an open netCDF file.

ncvarrename: C Interface

```
int ncvarrename(int ncid, int varid, const char* name);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID, returned from a previous call to `ncvardef` or `ncvarid`.

`name` New name for the specified variable.

Here is an example using `ncvarrename` to rename the variable `rh` to `rel_hum` in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;                        /* netCDF ID */
int  rh_id;                       /* variable ID */
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
ncredef(ncid);  /* put in define mode to rename variable */
rh_id = ncvarid (ncid, "rh");
ncvarrename (ncid, rh_id, "rel_hum");
ncendef(ncid);  /* leave define mode */
```

NCVREN: FORTRAN Interface

```
 SUBROUTINE NCVREN (INTEGER NCID, INTEGER VARID,
+                   CHARACTER*(*) NEWNAM, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID, returned from a previous call to `NCVDEF` or `NCVID`.

`NEWNAM` New name for the specified variable.

Here is an example using `NCVREN` to rename the variable `rh` to `rel_hum` in an existing netCDF file named '`foo.nc`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  NCID, RCODE
INTEGER  RHID              ! variable ID
   ...
NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
   ...
CALL NCREDF (CDFFID, RCODE)  ! enter definition mode
RHID = NCVID (NCID, 'rh', RCODE)  ! get ID
CALL NCVREN (NCID, RHID, 'rel_hum', RCODE)
CALL NCENDF (CDFFID, RCODE)  ! leave definition mode
```

7.15 Get Number of Bytes for a Data Type

The function `nctypelen` (or `NCTLEN` for FORTRAN) returns the number of bytes per netCDF data type.

In case of an error, `nctypelen` returns -1; `NCTLEN` returns a nonzero value in `rcode`. One possible cause of errors is:

- The specified data type is not a valid netCDF data type.

nctypelen: C Interface

```
int nctypelen (nc_type datatype);
```

`datatype` One of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`.

Here is an example using `nctypelen` to determine how many bytes are required to store a single value of the variable `rh` in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;        /* netCDF ID */
int  rh_id;       /* variable ID */
nc_type rh_type;           /* variable type */
int rh_ndims;              /* number of dims */
int rh_dims[MAX_VAR_DIMS]; /* variable shape */
int rh_natts;              /* number of attributes */
int rhbytes;      /* number of bytes per value for "rh" */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
/* get type.  we don't need name, since we already know it */
ncvarinq (ncid, rh_id, (char *) 0, &rh_type, &rh_ndims, rh_dims,
          &rh_natts);
rhbytes = nctypelen (rh_type);
```

NCTLEN: FORTRAN Interface

```
INTEGER FUNCTION NCTLEN (INTEGER TYPE ,INTEGER RCODE)
```

`TYPE` One of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCTLEN` to determine how many bytes are required to store a single value of the variable `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID              ! netCDF ID
      INTEGER  RHID               ! variable ID
      CHARACTER*31 RHNAME         ! variable name
      INTEGER  RHTYPE             ! variable type
      INTEGER  RHN                ! number of dimensions
      INTEGER  RHDIMS(MAXVDIMS)   ! variable shape
      INTEGER  RHNATT             ! number of attributes
      INTEGER  RHBYTS             ! bytes per value
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
* get type of "rh"
      CALL NCVINQ (NCID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT,
     +             RCODE)
      RHBYTS = NCTLEN (RHTYPE)
```

7.16 Get Information About Record Variables

The function ncrecinq returns information about the record variables (variables that use the unlimited dimension) in a netCDF file. The information returned is the number of record variables, their variable IDs, and the size (in bytes) for a record's worth of data for each record variable.

The ncrecinq function is not strictly necessary, since the information it returns can be computed from information returned by ncinquire, ncdiminq, and ncvarinq functions or their FORTRAN counterparts. This function is provided in the C interface for convenience only, to assist in using the C functions ncrecput and ncrecget.

In case of an error, ncrecinq returns -1. Possible causes of errors include:

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

ncrecinq: C Interface

```
int ncrecinq(int ncid, int* nrvars, int rvarids[], long rsizes[]);
```

`ncid` NetCDF ID, returned from a previous call to ncopen or nccreate.

`nrvars` Number of record variables.

`rvarids` Returned vector of nrvars variable IDs for the record variables in this netCDF file. The caller must allocate enough space for a vector of at least nrvars integers to be returned. The maximum possible number of variable IDs returned is given by the predefined constant MAX_NC_VARS. If this parameter is given as '(int *) 0', no vector will be returned so no space to hold the record variable IDs needs to be declared or allocated.

`rsizes` Returned vector of nrvars sizes for the record variables in this netCDF file. The size of a record variable is the number of bytes required to hold a record's worth of data, which is the product of the non-record dimensions and the size of data type, in bytes. The caller must allocate enough space for a vector of at least nrvars longs to be returned. The maximum possible number of variable IDs returned is given by the predefined constant MAX_NC_VARS. If this parameter is given as '(int *) 0', no vector will be returned so no space to hold the record variable sizes needs to be declared or allocated.

Here is an example using `ncrecinq` to find out about the record variables in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;                      /* netCDF ID */
int  nrvars;                    /* number of record variables */
int  rvarids[MAX_NC_VARS];      /* IDs of record variables */
long rvarsizes[MAX_NC_VARS];    /* record sizes of record variables */
   ...
ncid = ncopen ("foo.nc", NC_NOWRITE);
   ...
ncrecinq (ncid, &nrvars, rvarids, rvarsizes);
```

8 Attributes

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 file are defined when the file is first created, while the netCDF file 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, except in one case (`ncattname` or `NCANAM` in FORTRAN), where attributes are designated by variable ID and number because their names are unknown.

The attributes associated with a variable are typically defined right 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 assign attributes not associated with any variable to the netCDF as a whole, by using the `NC_GLOBAL` variable pseudo-ID. Global attributes may be used for purposes such as providing a title or processing history for a netCDF data set.

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 to another.
- Get name of attribute from its number.
- Rename an attribute.
- Delete an attribute.

8.1 Create an Attribute

The function `ncattput` (or `NCAPT` or `NCAPTC` for FORTRAN) adds or changes a variable attribute or global attribute of an open netCDF file. If this attribute is new, or if the space required to store the attribute is greater than before, the netCDF file must be in define mode.

In case of an error, `ncattput` returns -1; `NCAPT` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified netCDF type is invalid.
- The specified length is negative.
- The specified open netCDF file is in data mode and the specified attribute would expand.
- The specified open netCDF file is in data mode and the specified attribute does not already exist.
- The specified netCDF ID does not refer to an open netCDF file.

ncattput: C Interface

```
int ncattput(int ncid, int varid, const char* name, nc_type datatype,
             int len, const void* values);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID of the variable to which the attribute will be assigned or `NC_GLOBAL` for a global attribute.

`name` Attribute name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('`_`'). Case is significant. 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 ⟨undefined⟩ [Attribute Conventions], page ⟨undefined⟩, for examples of attribute conventions.

`datatype` One of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`.

`len` Number of values provided for the attribute. If the attribute is of type `NC_CHAR`, this is one more than the string length (since the terminating null character is stored).

`values` Pointer to one or more data values. The pointer is declared to be of the type `void *` because it can point to data of any of the basic netCDF types. The data should be of the appropriate type for the netCDF attribute. **Warning: neither the compiler nor the netCDF software can detect whether the wrong type of data is used.**

Here is an example using `ncattput` to add a variable attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` to an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;                         /* netCDF ID */
int  rh_id;                        /* variable ID */
static double rh_range[] = {0.0, 100.0};  /* attribute vals */
static char title[] = "example netCDF file";
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
ncredef(ncid);                    /* enter define mode */
rh_id = ncvarid (ncid, "rh");
   ...
ncattput (ncid, rh_id, "valid_range", NC_DOUBLE, 2, rh_range);
ncattput (ncid, NC_GLOBAL, "title", NC_CHAR, strlen(title)+1,
          title);
   ...
ncendef(ncid);                    /* leave define mode */
```

NCAPT, NCAPTC: FORTRAN Interface

```
      SUBROUTINE NCAPT (INTEGER NCID, INTEGER VARID,
     +                  CHARACTER*(*) ATTNAM, INTEGER ATTYPE,
     +                  INTEGER ATTLEN, type VALUE,
     +                  INTEGER RCODE)

      SUBROUTINE NCAPTC (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) ATTNAM, INTEGER ATTYPE,
     +                   INTEGER LENSTR, CHARACTER*(*) STRING,
     +                   INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCAPT` and `NCAPTC`, for creating attributes. The first is for attributes of numeric type, and the second is for attributes of character-string type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID, returned from a previous call to `NCVDEF` or `NCVID`.

`ATTNAM` Attribute name. Must begin with an alphabetic character, followed by zero or more alphanumeric characters including the underscore ('_'). Case is significant. 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. A table of conventional attribute names is presented in the earlier chapter on the netCDF interface.

`ATTYPE` One of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`.

`ATTLEN` In `NCAPT`, the number of numeric values provided for the attribute.

`VALUE` In `NCAPT`, an array of `ATTLEN` data values. The data should be of the appropriate type for the netCDF attribute. **Warning: neither the compiler nor the netCDF software can detect if the wrong type of data is used.**

`STRING` In `NCAPTC`, the character-string value of the attribute.

`LENSTR` In `NCAPTC`, the total declared length (in characters) of the `STRING` parameter. Note that this is not necessarily the same as the value returned by the FORTRAN `LEN` function, because an array argument may be provided.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCAPT` to add a variable attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` to an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID, RCODE
      INTEGER  RHID               ! variable ID
      DOUBLE RHRNGE(2)
      DATA RHRNGE /0.0D0, 100.0D0/
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      CALL NCREDF (NCID, RCODE)     ! enter define mode
      RHID = NCVID (NCID, 'rh', RCODE)! get ID
         ...
      CALL NCAPT (NCID, RHID, 'valid_range', NCDOUBLE, 2,
     +            RHRNGE, RCODE)
      CALL NCAPTC (NCID, NCGLOBAL, 'title', NCCHAR, 19,
     +            'example netCDF file', RCODE)
         ...
      CALL NCENDF (NCID, RCODE)     ! leave define mode
```

8.2 Get Information about an Attribute

The function `ncattinq` (or `NCAINQ` for FORTRAN) returns information about a netCDF attribute, given its variable ID and name. The information returned is the type and length of the attribute.

In case of an error, `ncattinq` returns -1; `NCAINQ` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

ncattinq: C Interface

```
int ncattinq(int ncid, int varid, const char* name,
             nc_type* datatype, int* len);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID of the attribute's variable, or `NC_GLOBAL` for a global attribute.

`name` Attribute name.

`datatype` Returned attribute type, one of the set of predefined netCDF data types. The type of this parameter, `nc_type`, is defined in the netCDF header file. The valid netCDF data types are `NC_BYTE`, `NC_CHAR`, `NC_SHORT`, `NC_LONG`, `NC_FLOAT`, and `NC_DOUBLE`. If this parameter is given as '`(nc_type *) 0`', no type will be returned so no variable to hold the type needs to be declared.

`len` Returned number of values currently stored in the attribute. If the attribute is of type `NC_CHAR`, this is one more than the string length (since the terminating null character is stored). If this parameter is given as '`(int *) 0`', no length will be returned so no variable to hold this information needs to be declared.

Here is an example using `ncattinq` 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 file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;                 /* netCDF ID */
int  rh_id;                /* variable ID */
nc_type vr_type, t_type;   /* attribute types */
int  vr_len, t_len;        /* attribute lengths *'

   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
ncattinq (ncid, rh_id, "valid_range", &vr_type, &vr_len);
ncattinq (ncid, NC_GLOBAL, "title", &t_type, &t_len);
   ...
```

NCAINQ: FORTRAN Interface

```
      SUBROUTINE NCAINQ (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) ATTNAM, INTEGER ATTYPE,
     +                   INTEGER ATTLEN,INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID of the attribute's variable, or `NCGLOBAL` for a global attribute.

`ATTNAM` Attribute name.

`ATTYPE` Returned attribute type, one of the set of predefined netCDF data types. The valid netCDF data types are `NCBYTE`, `NCCHAR`, `NCSHORT`, `NCLONG`, `NCFLOAT`, and `NCDOUBLE`.

`ATTLEN` Returned number of values currently stored in the attribute. For a string-valued attribute, this is the number of characters in the string.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCAINQ` to add a variable attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` to an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID, RCODE
      INTEGER  RHID               ! variable ID
      INTEGER  VRTYPE, TTYPE      ! attribute types
      INTEGER  VRLEN, TLEN        ! attribute lengths
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)! get ID
         ...
      CALL NCAINQ (NCID, RHID, 'valid_range', VRTYPE, VRLEN,
     +             RCODE)
      CALL NCAINQ (NCID, NCGLOBAL, 'title', TTYPE, TLEN,
     +             RCODE)
```

8.3 Get Attribute's Values

The function `ncattget` (or `NCAGT` or `NCAGTC` for FORTRAN) gets the value(s) of a netCDF attribute, given its variable ID and name.

In case of an error, `ncattget` returns -1; `NCAGT` returns a nonzero value in `rcode`. Possible causes of errors include:

- The variable ID is invalid for the specified netCDF file.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

ncattget: C Interface

```
int ncattget(int ncid, int varid, const char* name, void* value);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` Variable ID of the attribute's variable, or `NC_GLOBAL` for a global attribute.

`name` Attribute name.

`value` Returned attribute values. All elements of the vector of attribute values are returned, so you must allocate enough space to hold them. If you don't know how much space to reserve, call `ncattinq` first to find out the length of the attribute.

Here is an example using `ncattget` 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 file 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.h"
   ...
int  ncid;                  /* netCDF ID */
int  rh_id;                 /* variable ID */
nc_type vr_type, t_type;    /* attribute types */
int  vr_len, t_len;         /* attribute lengths */
double *vr_val;             /* ptr to attribute values */
char *title;                /* ptr to attribute values */
extern char *malloc();      /* memory allocator */

   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* find out how much space is needed for attribute values */
ncattinq (ncid, rh_id, "valid_range", &vr_type, &vr_len);
ncattinq (ncid, NC_GLOBAL, "title", &t_type, &t_len);

/* allocate required space before retrieving values */
vr_val = (double *) malloc(vr_len * nctypelen(vr_type));
title = (char *) malloc(t_len * nctypelen(t_type));

/* get attribute values */
ncattget(ncid, rh_id, "valid_range", (void *)vr_val);
ncattget(ncid, NC_GLOBAL, "title", (void *)title);
   ...
```

NCAGT, NCAGTC: FORTRAN Interface

```
      SUBROUTINE NCAGT (INTEGER NCID, INTEGER VARID,
     +                  CHARACTER*(*) ATTNAM, type VALUES,
     +                  INTEGER RCODE)

      SUBROUTINE NCAGTC (INTEGER NCID, INTEGER VARID,
     +                  CHARACTER*(*) ATTNAM, CHARACTER*(*) STRING,
     +                  INTEGER LENSTR, INTEGER RCODE)
```

There are two FORTRAN subroutines, `NCAGT` and `NCAGTC`, for retrieving attribute values. The first is for attributes of numeric type, and the second is for attributes of character-string type.

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` Variable ID of the attribute's variable, or `NCGLOBAL` for a global attribute.

`ATTNAM` Attribute name.

`VALUES` 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 `NCAINQ` first to find out the length of the attribute. **Warning: neither the compiler nor the netCDF software can detect if the wrong type of data is used.**

`STRING` In `NCAGTC`, the character-string value of the attribute.

`LENSTR` In `NCAGTC`, the total declared length (in characters) of the `STRING` parameter in the caller. Note that this is not necessarily the same as the value returned by the FORTRAN `LEN` function, because an array argument may be provided. `NCAGTC` will check to make sure the requested data will fit in `LENSTR` characters.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCAGT` to determine the values of an attribute named `valid_range` for a netCDF variable named `rh` and a global attribute named `title` in an existing netCDF file named '`foo.nc`'. In this example, it is assumed that we don't know how many values will be returned, so we first inquire about the length of the attributes to make sure we have enough space to store them:

```
      INCLUDE 'netcdf.inc'
         ...
      PARAMETER (MVRLEN=3) ! max number of "valid_range" values
      PARAMETER (MTLEN=80) ! max length of "title" attribute
      INTEGER  NCID, RCODE
      INTEGER  RHID               ! variable ID
      INTEGER  VRTYPE, TTYPE      ! attribute types
      INTEGER  VRLEN, TLEN        ! attribute lengths
      DOUBLE PRECISION VRVAL(MVRLEN) ! vr attribute values
      CHARACTER*80 TITLE          ! title attribute values
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE) ! get ID
         ...
* find out attribute lengths, to make sure we have enough space
      CALL NCAINQ (NCID, RHID, 'valid_range', VRTYPE, VRLEN,
     +             RCODE)
      CALL NCAINQ (NCID, NCGLOBAL, 'title', TTYPE, TLEN,
     +             RCODE)
* get attribute values, if not too big
      IF (VRLEN > MVRLEN) THEN
          WRITE (*,*) 'valid_range attribute too big!'
          CALL EXIT
      ELSE
          CALL NCAGT (NCID, RHID, 'valid_range', VRVAL, RCODE)
      ENDIF
      IF (TLEN > MTLEN) THEN
          WRITE (*,*) 'title attribute too big!'
          CALL EXIT
      ELSE
          CALL NCAGTC (NCID, NCGLOBAL, 'title', TITLE, MTLEN, RCODE)
      ENDIF
```

8.4 Copy Attribute from One NetCDF to Another

The function `ncattcopy` (or `NCACPY` for FORTRAN) copies an attribute from one open netCDF file to another. It can also be used to copy an attribute from one variable to another within the same netCDF.

In case of an error, `ncattcopy` returns -1; `NCACPY` returns a nonzero value in `rcode`. Possible causes of errors include:

- The input or output variable ID is invalid for the specified netCDF file.
- The specified attribute does not exist.
- The output netCDF is not in define mode and the attribute is new for the output file is larger than the existing attribute.
- The input or output netCDF ID does not refer to an open netCDF file.

ncattcopy: C Interface

```
int ncattcopy(int incdf, int invar, const char* name, int outcdf, int outvar);
```

`incdf` The netCDF ID of an input netCDF file from which the attribute will be copied, returned from a previous call to `ncopen` or `nccreate`.

`invar` ID of the variable in the input netCDF file from which the attribute will be copied, or `NC_GLOBAL` for a global attribute.

`name` Name of the attribute in the input netCDF file to be copied.

`outcdf` The netCDF ID of the output netCDF file to which the attribute will be copied, returned from a previous call to `ncopen` or `nccreate`. It is permissible for the input and output netCDF IDs to be the same. The output netCDF file 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.

`outvar` ID of the variable in the output netCDF file to which the attribute will be copied, or `NC_GLOBAL` to copy to a global attribute.

Here is an example using `ncattcopy` to copy the variable attribute `units` from the variable `rh` in an existing netCDF file named '`foo.nc`' to the variable `avgrh` in another existing netCDF file named '`bar.nc`', assuming that the variable `avgrh` already exists, but does not yet have a `units` attribute:

```
#include "netcdf.h"
   ...
int  ncid1, ncid2;       /* netCDF IDs */
int  rh_id, avgrh_id;      /* variable IDs */
   ...
ncid1 = ncopen("foo.nc", NC_NOWRITE);
ncid2 = ncopen("bar.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid1, "rh");
avgrh_id = ncvarid (ncid2, "avgrh");
   ...
ncredef(ncid2);           /* enter define mode */
/* copy variable attribute from "rh" to "avgrh" */
ncattcopy(ncid1, rh_id, "units", ncid2, avgrh_id);
   ...
ncendef(ncid2);           /* leave define mode */
```

NCACPY: FORTRAN Interface

```
      SUBROUTINE NCACPY (INTEGER INCDF, INTEGER INVAR,
     +                   CHARACTER*(*) ATTNAM, INTEGER OUTCDF,
     +                   INTEGER OUTVAR, INTEGER RCODE)
```

`INCDF` The netCDF ID of an input netCDF file from which the attribute will be copied, returned from a previous call to `NCOPN` or `NCCRE`.

`INVAR` ID of the variable in the input netCDF file from which the attribute will be copied, or `NCGLOBAL` for a global attribute.

`ATTNAM` Name of the attribute in the input netCDF file to be copied.

`OUTCDF` The netCDF ID of the output netCDF file to which the attribute will be copied, returned from a previous call to `NCOPN` or `NCCRE`. It is permissible for the input and output netCDF IDs to be the same. The output netCDF file 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.

`OUTVAR` ID of the variable in the output netCDF file to which the attribute will be copied, or `NCGLOBAL` to copy to a global attribute.

Here is an example using `NCACPY` to copy the variable attribute `units` from the variable `rh` in an existing netCDF file named '`foo.nc`' to the variable `avgrh` in another existing netCDF file named '`bar.nc`', assuming that the variable `avgrh` already exists, but does not yet have a `units` attribute:

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID1, NCID2     ! netCDF IDs
      INTEGER  RHID, AVRHID       ! variable IDs
         ...
      NCID1 = NCOPN ('foo.nc', NCNOWRIT, RCODE)
      NCID2 = NCOPN ('bar.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID1, 'rh', RCODE)
      AVRHID = NCVID (NCID2, 'avgrh', RCODE)
         ...
      CALL NCREDF (NCID2, RCODE)  ! enter define mode
* copy variable attribute from "rh" to "avgrh"
      CALL NCACPY (NCID1, RHID, 'units', NCID2, AVRHID, RCODE)
         ...
      CALL NCENDF (NCID2, RCODE)  ! leave define mode
```

8.5 Get Name of Attribute from Its Number

The function `ncattname` (or `NCANAM` for FORTRAN) 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.

In case of an error, `ncattname` returns -1; `NCANAM` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified variable ID is not valid.
- The specified attribute number is negative or more than the number of attributes defined for the specified variable.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

ncattname: C Interface

```
int ncattname (int ncid, int varid, int attnum, char* name);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` ID of the attribute's variable, or `NC_GLOBAL` for a global attribute.

`attnum` Number of the attribute. The attributes for each variable are numbered from 0 (the first attribute) to `nvatts-1`, where `nvatts` is the number of attributes for the variable, as returned from a call to `ncvarinq`.

`name` Returned attribute name. The caller must allocate space for the returned name. The maximum possible length, in characters, of an attribute name is given by the predefined constant `MAX_NC_NAME`. If the name parameter is given as `(char *) 0`, no name will be returned and no space needs to be allocated.

Here is an example using `ncattname` to determine the name of the first attribute of the variable `rh` in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;        /* netCDF ID */
int  rh_id;       /* variable ID */
char attname[MAX_NC_NAME];  /* maximum-size attribute name */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* get name of first attribute (number 0) */
ncattname(ncid, rh_id, 0, attname);
```

NCANAM: FORTRAN Interface

```
      SUBROUTINE NCANAM (INTEGER NCID, INTEGER VARID,
     +                   INTEGER ATTNUM, CHARACTER*(*) ATTNAM,
     +                   INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` ID of the attribute's variable, or `NCGLOBAL` for a global attribute.

`ATTNUM` Number of the attribute. The attributes for each variable are numbered from 1 (the first attribute) to `NVATTS`, where `NVATTS` is the number of attributes for the variable, as returned from a call to `NCVINQ`.

`ATTNAM` Returned attribute name. The caller must allocate space for the returned name. The maximum possible length, in characters, of an attribute name is given by the predefined constant `MAXNCNAM`.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCANAM` determine the name of the first attribute of the variable `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID     ! netCDF ID
      INTEGER  RHID      ! variable ID
* 31 in the following should be MAXNCNAM
      CHARACTER*31 ATTNAM
         ...
      NCID = NCOPN ('foo.nc', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
         ...
* get name of first attribute (number 1)
      CALL NCANAM (NCID, RHID, 1, ATTNAM, RCODE)
```

8.6 Rename an Attribute

The function `ncattrename` (or `NCAREN` for FORTRAN) changes the name of an attribute. If the new name is longer than the original name, the netCDF must be in define mode. You cannot rename an attribute to have the same name as another attribute of the same variable.

In case of an error, `ncattrename` returns -1; `NCAREN` returns a nonzero value in `rcode`. 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 file 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 file.

ncattrename: C Interface

```
int ncattrename (int ncid, int varid, const char* name, const char* newname);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`

`varid` ID of the attribute's variable, or `NC_GLOBAL` for a global attribute

`name` The original attribute name.

`newname` The new name to be assigned to the specified attribute. If the new name is longer than the old name, the netCDF file must be in define mode.

Here is an example using `ncattrename` to rename the variable attribute `units` to `Units` for a variable `rh` in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;        /* netCDF ID */
int  rh_id;       /* variable id */
   ...
ncid = ncopen("foo.nc", NC_NOWRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* rename attribute */
ncattrename(ncid, rh_id, "units", "Units");
```

NCAREN: FORTRAN Interface

```
      SUBROUTINE NCAREN (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) ATTNAM,
     +                   CHARACTER*(*) NEWNAM, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`

`VARID` ID of the attribute's variable, or `NCGLOBAL` for a global attribute

`ATTNAM` The original attribute name.

`NEWNAM` The new name to be assigned to the specified attribute. If the new name is longer than the old name, the netCDF file must be in define mode.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCAREN` to rename the variable attribute `units` to `Units` for a variable `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE "netcdf.inc"
         ...
      INTEGER  NCID     ! netCDF ID
      INTEGER  RHID      ! variable ID
         ...
      NCID = NCOPN ("foo.nc", NCNOWRIT, RCODE)
         ...
      RHID = NCVID (NCID, "rh", RCODE)
         ...
* rename attribute
      CALL NCAREN (NCID, RHID, "units", "Units", RCODE)
```

8.7 Delete an Attribute

The function `ncattdel` (or `NCADEL` for FORTRAN) deletes a netCDF attribute from an open netCDF file. The netCDF file must be in define mode.

In case of an error, `ncattdel` returns -1; `NCADEL` returns a nonzero value in `rcode`. Possible causes of errors include:

- The specified variable ID is not valid.
- The specified netCDF file is in data mode.
- The specified attribute does not exist.
- The specified netCDF ID does not refer to an open netCDF file.

ncattdel: C Interface

```
int ncattdel (int ncid, int varid, const char* name);
```

`ncid` NetCDF ID, returned from a previous call to `ncopen` or `nccreate`.

`varid` ID of the attribute's variable, or `NC_GLOBAL` for a global attribute.

`name` The name of the attribute to be deleted.

Here is an example using `ncattdel` to delete the variable attribute `Units` for a variable `rh` in an existing netCDF file named '`foo.nc`':

```
#include "netcdf.h"
   ...
int  ncid;        /* netCDF ID */
int  rh_id;       /* variable ID */
   ...
ncid = ncopen("foo.nc", NC_WRITE);
   ...
rh_id = ncvarid (ncid, "rh");
   ...
/* delete attribute */
ncredef(ncid);                  /* enter define mode */
ncattdel(ncid, rh_id, "Units");
ncendef(ncid);                  /* leave define mode */
```

NCADEL: FORTRAN Interface

```
      SUBROUTINE NCADEL (INTEGER NCID, INTEGER VARID,
     +                   CHARACTER*(*) ATTNAM, INTEGER RCODE)
```

`NCID` NetCDF ID, returned from a previous call to `NCOPN` or `NCCRE`.

`VARID` ID of the attribute's variable, or `NCGLOBAL` for a global attribute.

`ATTNAM` The original attribute name.

`RCODE` Returned error code. If no errors occurred, 0 is returned.

Here is an example using `NCADEL` to delete the variable attribute `Units` for a variable `rh` in an existing netCDF file named '`foo.nc`':

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  NCID     ! netCDF ID
      INTEGER  RHID      ! variable ID
         ...
      NCID = NCOPN ('foo.nc', NCWRITE, RCODE)
         ...
      RHID = NCVID (NCID, 'rh', RCODE)
         ...
* delete attribute
      CALL NCREDF (NCID, RCODE)  ! enter define mode
      CALL NCADEL (NCID, RHID, 'Units', RCODE)
      CALL NCENDF (NCID, RCODE)  ! leave define mode
```

9 NetCDF File Structure and Performance

NetCDF is a data abstraction for scientific data access and a software library that provides a concrete implementation of the interfaces that support that abstraction. The implementation provides a machine-independent format for representing scientific data. Although the netCDF file format is completely hidden below the interfaces, some understanding of the implementation and associated file structure may help to make clear which netCDF operations are expensive and why.

A detailed description of the netCDF *format* is not appropriate for this User's Guide, however. It is not needed to read and write netCDF files or understand efficiency issues. Programs that use only the documented interfaces and that make no other assumptions about the format will continue to work even if the netCDF format is changed in the future, because any such change will be made below the documented interfaces and will support earlier versions of netCDF data.

This chapter describes the structure of a netCDF file and some characteristics of the XDR layer that provides network transparency in enough detail to understand netCDF performance issues.

9.1 Parts of a NetCDF File

A netCDF data set is stored as a single file comprising three parts:

- a *header*, containing all the information about dimensions, attributes, and variables except for the variable data;
- *fixed-size data*, containing the data for variables that don't have an unlimited dimension; and
- *record data*, containing the data records for variables that have an unlimited dimension.

All the data are represented in XDR form to make them machine-independent.

The descriptive header at the beginning of the netCDF file is an XDR encoding of a high-level data structure that represents information about the dimensions, variables, and attributes in the file. The variable descriptions in this header contain offsets to the beginning of each variable's data or the relative offset of a variable within a record. The descriptions also contain the dimension size and information needed to determine how to map multidimensional indices for each variable to the appropriate offsets.

This header has no usable extra space; it is only as large as it needs to be for the dimensions, variables, and attributes in each netCDF file. This has the advantage that netCDF files are compact, requiring very little overhead to store the ancillary data that makes the files self-describing. A potential disadvantage of this organization is that any operation on a netCDF file that requires expanding the header, for example adding a set of new dimensions and new variables to an existing netCDF file, will be as expensive as copying the file. This expense is incurred when `ncendef()` is called, after a call to `ncredef()`. If you create all necessary dimensions, variables, and attributes *before* writing variable data, and avoid later additions and renamings of netCDF components that require more space in the header part of the file, you avoid the cost associated with expanding the header.

The fixed-size data part that follows the header contains all the variable data for variables that do not employ the unlimited (record) dimension. The data for each variable is stored contiguously in this part of the file. If there is no unlimited dimension, this is the last part of the netCDF file.

The record-data part that follows the fixed-size data consists of a variable number of records, each of which contains data for all the record variables. The record data for each variable is stored contiguously in each record.

The order in which the data in the fixed-size data part and in each record appears is the same as the order in which the variables were defined, in increasing numerical order by netCDF variable ID. This knowledge can sometimes be used to enhance data access performance, since the best data access is currently achieved by reading or writing the data in sequential order.

9.2 The XDR Layer

XDR is a standard for describing and encoding data and a library of functions for external data representation, allowing programmers to encode data structures in a machine-independent way. NetCDF employs XDR for representing all data, in both the header part and the data parts. XDR is used to write portable data that can be read on any other machine for which the XDR library has been implemented.

Many vendors provide an XDR library along with other C run-time libraries. The netCDF software distribution also includes Sun's portable implementation of XDR for platforms that don't already have a vendor-supplied XDR library.

An I/O layer implemented much like the C standard I/O (stdio) library is used by the XDR layer to read and write XDR-encoded data to netCDF files. Hence an understanding of the standard I/O library provides answers to most questions about multiple processes accessing data concurrently, the use of I/O buffers, and the costs of opening and closing netCDF files. In particular, it is possible to have one process writing a netCDF file while other processes read it. Data reads and writes are no more atomic than calls to stdio `fread()` and `fwrite()`. An `ncsync()` call (`NCSNC()` for FORTRAN) is analogous to the `fflush()` call in the standard I/O library, writing unwritten buffered data so other processes can read it; `ncsync()` also brings header changes up-to-date (e.g., changes to attribute values).

As in the stdio library, flushes are also performed when "seeks" occur to a different area of the file. Hence the order of read and write operations can influence I/O performance significantly. Reading data in the same order in which it was written within each record will minimize buffer flushes.

There is one unusual case where the situation is more complex: when a writer enters define mode to add some additional dimensions, variables, or attributes to an existing netCDF file that is also open for reading by other processes. In this case, when the writer leaves define mode, a new copy of the file is created with the new dimensions, attributes, or variables and the old data, but readers that still have the file open will not see the changes. You should not expect netCDF data access to work with multiple writers having the same file open for writing simultaneously.

For VMS systems, the performance penalty for permitting shared access (under the current implementation of stdio in the C run-time library) seemed too great to make shared access the default, so netCDF files on VMS are opened non-shared. This still permits multiple simultaneous readers of the same file, but one writer prevents any readers from accessing the file. Implementors can easily allow shared access for a VMS implementation, if shared access is a more important requirement than access speed.

It is possible to tune an implementation of netCDF for some platforms by replacing the I/O layer beneath XDR with a different platform-specific I/O layer. This may change the similarities between netCDF and standard I/O, and hence characteristics related to data sharing, buffering, and the cost of I/O operations.

The cost of using a canonical representation for data like XDR varies according to the type of data and whether the XDR form is the same as the machine's native form for that type. XDR is especially efficient for byte, character, and short integer data.

For some data types on some machines, the time required to convert data to and from XDR form can be significant. The best case is byte arrays, for which very little conversion expense occurs, since the XDR library has built-in support for them. The netCDF implementation includes similar support added to XDR for arrays of short (16-bit) integers. The worst case is reading or writing large arrays of floating-point data on a machine that does not use IEEE floating-point as its native representation. The XDR library incurs the expense of a function call for each floating-point quantity accessed. On some architectures the cost of a function invocation for each floating-point number can dominate the cost of netCDF access to floating-point fields.

The distributed netCDF implementation is meant to be portable. Platform-specific ports that further optimize the implementation for better I/O performance or that unroll the loops in the XDR library to optimize XDR conversion of long integer and floating-point arrays are practical and desirable in cases where higher performance for data access is necessary.

10 Higher-Level netCDF Operations

One of the primary reasons for using the netCDF interface for both scientific data and applications that deal with scientific data is to take advantage of higher-level netCDF operations and generic applications for processing netCDF files. NetCDF *operators* read one or more netCDF input files and write a netCDF output file. NetCDF *utilities* read one or more netCDF files and produce some other kind of output (e.g. a text file or graphics), or read some other kind of input (e.g. a text file) and write netCDF files. Currently two netCDF utilities are available as part of the netCDF software distribution: `ncgen` and `ncdump`. A preliminary set of example netCDF operators is available as a separate package.

This chapter describes the `ncgen` and `ncdump` utilities. These two tools convert between binary netCDF files and an ASCII representation of netCDF files. The output of `ncdump` and the input to `ncgen` is an ASCII description of a netCDF file in a tiny language known as CDL (network Common data form Description Language).

The preliminary set of example netCDF operators is available via anonymous FTP of the file `pub/sdm/ncopers<version>.tar.Z` from host `unidata.ucar.edu`. The programs include: `ncextr`, which extracts a subset of variables from a netCDF file into an output netCDF file; `ncstat`, which reads a netCDF file and writes selected statistics along one or more summary dimensions into another netCDF file; and `ncbarne`, which performs a Barnes objective analysis of raw data contained in a netCDF file and writes the result to an output netCDF file. These programs are documented in the ncopers release.

A number of freely-available and commercial software packages are also available for manipulating or graphically displaying netCDF data. A list of these packages is maintained in the file `pub/netcdf/utilities.txt` from host `unidata.ucar.edu`.

10.1 CDL Syntax

Below is an example of CDL, describing a netCDF file with several named dimensions (`lat`, `lon`, `time`), variables (`z`, `t`, `p`, `rh`, `lat`, `lon`, `time`), variable attributes (`units`, `valid_range`, `_FillValue`), and some data.

```
netcdf foo {    // example netCDF specification in CDL

dimensions:
lat = 10, lon = 5, time = unlimited ;

variables:
  long    lat(lat), lon(lon), time(time);
  float   z(time,lat,lon), t(time,lat,lon);
  double  p(time,lat,lon);
  long    rh(time,lat,lon);

  lat:units = "degrees_north";
  lon:units = "degrees_east";
  time:units = "seconds";
  z:units = "meters";
  z:valid_range = 0., 5000.;
  p:_FillValue = -9999.;
  rh:_FillValue = -1;

data:
  lat   = 0, 10, 20, 30, 40, 50, 60, 70, 80, 90;
  lon   = -140, -118, -96, -84, -52;
}
```

All CDL statements are terminated by a semicolon. Spaces, tabs, and newlines can be used freely for readability. Comments may follow the double slash characters `//` on any line.

A CDL description consists of three optional parts: dimensions, variables, and data. The variable part may contain variable declarations and attribute assignments.

A dimension is used to define the shape of one or more of the multidimensional variables described by the CDL description. A dimension has a name and a size. At most one dimension in a CDL description can have the unlimited size, which means a variable using this dimension can grow to any length (like a record number in a file).

A variable represents a multidimensional array of values of the same type. A variable has a name, a data type, and a shape described by its list of dimensions. Each variable may also have associated attributes (see below) as well as data values. The name, data type, and shape of a variable are specified by its declaration in the variable section of a CDL description. A variable may have the same name as a dimension; by convention such a variable is one-dimensional and contains coordinates of the dimension it names. Dimensions need not have corresponding variables.

An attribute contains information about a variable or about the whole netCDF data set. Attributes are used to specify such properties as units, special values, maximum and minimum valid values, scaling factors, offsets, and parameters. Attribute information is represented by single values or arrays of values. For example, `units` is an attribute represented by a character array such as `celsius`. An attribute has an associated variable, a name, a data type, a length, and a value. In contrast to variables that are intended for data, attributes are intended for ancillary data (data about data).

In CDL, an attribute is designated by a variable and attribute name, separated by a colon ('`:`'). It is possible to assign global attributes not associated with any variable to the netCDF file as a whole by using the colon ('`:`') before the attribute name. The data type of an attribute in CDL is derived from the type of the value assigned to it. The length of an attribute is the number of data values or the number of characters in the character string assigned to it. Multiple values are assigned to noncharacter attributes by separating the values with commas ('`,`'). All values assigned to an attribute must be of the same type.

CDL names for variables, attributes, and dimensions may be any combination of alphabetic or numeric characters as well as '`_`' and '`-`' characters, but names beginning with '`_`' are reserved for use by the library. Case is significant in CDL names. The netCDF library does not enforce any restrictions on netCDF names, so it is possible (though unwise) to define variables with names that are not valid CDL names. The names for the primitive data types are reserved words in CDL, so the names of variables, dimensions, and attributes must not be type names.

The optional data section of a CDL description is where netCDF variables may be initialized. The syntax of an initialization is simple:

variable = *value_1*, *value_2*, ...;

The comma-delimited list of constants may be separated by spaces, tabs, and newlines. For multidimensional arrays, the last dimension varies fastest. Thus, row-order rather than column order is used for matrices. If fewer values are supplied than are needed to fill a variable, it is extended with a type-dependent fill value. The types of constants need not match the type declared for a variable; coercions are done to convert integers to floating point, for example. All meaningful type conversions are supported.

10.2 CDL Data Types

The CDL data types are:

`char` Characters.

`byte` Eight-bit data, including zero bytes.

`short` 16-bit signed integers.

`long` 32-bit signed integers.

`int` (Synonymous with long).

`float` IEEE single-precision floating point (32 bits).

`real` (Synonymous with float).

`double` IEEE double-precision floating point (64 bits).

Except for the added data-type `byte` and the lack of the type qualifier `unsigned`, CDL supports the same primitive data types as C. In declarations, type names may be specified in either upper or lower case.

`Bytes` differ from characters in that they are intended to hold eight bits of data, and the zero byte has no special significance, as it may for character data. `Ncgen` converts `byte` declarations to `char` declarations in the output C code and to the nonstandard `BYTE` declaration (or similar platform-specific declaration) in output FORTRAN code.

`Shorts` can hold values between -32768 and 32767. `Ncgen` converts `short` declarations to `short` declarations in the output C code and to the nonstandard `INTEGER*2` declaration in output FORTRAN code.

`Longs` can hold values between -2147483648 and 2147483647. `Ncgen` converts `long` declarations to `long` declarations in the output C code and to `INTEGER` declarations in output FORTRAN code. In CDL declarations `int` and `integer` are accepted as synonyms for `long`.

`Floats` can hold values between about -3.4+38 and 3.4+38. Their external representation is as 32-bit IEEE normalized single-precision floating-point numbers. `Ncgen` converts `float` declarations to `float` declarations in the output C code and to `REAL` declarations in output FORTRAN code. In CDL declarations `real` is accepted as a synonym for `float`.

`Doubles` can hold values between about -1.7+308 and 1.7+308. Their external representation is as 64-bit IEEE standard normalized double-precision, floating-point numbers. `Ncgen` converts `double` declarations to double declarations in the output C code and to `DOUBLE PRECISION` declarations in output FORTRAN code.

10.3 CDL Notation for Data Constants

This section explains the CDL notation for netCDF constants.

Attributes are initialized in the `variables` section of a CDL description by providing a list of constants that determines the attribute's type and length. (In the C and FORTRAN procedural interfaces to the netCDF library, the type and length of an attribute must be explicitly provided when it is defined.) Since neither C nor FORTRAN provide suitable standard syntax to distinguish between constants of type `byte` and `char`, `short` and `long`, or `float` and `double` (except that FORTRAN provides the latter), CDL defines a syntax for constant values that allows it to determine the netCDF type of any constant. The syntax for CDL constants is similar to C syntax, except that type suffixes are appended to `short`s and `float`s to distinguish them from `long`s and `double`s.

A `byte` constant is represented by a single character or multiple character escape sequence enclosed in single quotes. For example:

```
'a'     // ASCII a
'\0'    // a zero byte
'\n'    // ASCII newline character
'\33'   // ASCII escape character (33 octal)
'\x2b'  // ASCII plus (2b hex)
'\377'  // 377 octal = 255 decimal, a non-ASCII byte
```

Character constants are enclosed in double quotes. A character array may be represented as a string enclosed in double quotes. Multiple strings are concatenated into a single array of characters, permitting long character arrays to appear on multiple lines. To support multiple variable-length string values, a conventional delimiter such as '`,`' may be used, but interpretation of any such convention for a string delimiter must be implemented in software above the netCDF library layer. The usual escape conventions for C strings are honored. For example:

```
"a"              // ASCII 'a'
"Two\nlines\n"   // a 10-character string with two embedded newlines
"a bell:\007"    // a string containing an ASCII bell
"ab","cde"       // the same as "abcde"
```

The form of a `short` constant is an integer constant with an '`s`' or '`S`' appended. If a `short` constant begins with '`0`', it is interpreted as octal. When it begins with `0x`, it is interpreted as a hexadecimal constant. For example:

```
2s      // a short 2
0123s   // octal
0x7ffs  // hexadecimal
```

The form of a `long` constant is an ordinary integer constant, although it is acceptable to append an optional '`l`' or '`L`'. If a `long` constant begins with '`0`', it is interpreted as octal. When it begins with `0x`, it is interpreted as a hexadecimal constant. Examples of valid `long` constants include:

```
-2
1234567890L
0123            // octal
0x7ff           // hexadecimal
```

The `float` type is appropriate for representing data with about seven significant digits of precision. The form of a `float` constant is the same as a C floating-point constant with an '`f`' or '`F`' appended. A decimal point is required in a CDL `float` to distinguish it from an integer. For example, the following are all acceptable `float` constants:

```
-2.0f
3.14159265358979f       // will be truncated to less precision
1.f
.1f
```

The `double` type is appropriate for representing floating-point data with about 16 significant digits of precision. The form of a `double` constant is the same as a C floating-point constant. An optional '`d`' or '`D`' may be appended. A decimal point is required in a CDL `double` to distinguish it from an `integer`. For example, the following are all acceptable double constants:

```
-2.0
3.141592653589793
1.0e-20
1.d
```

10.4 ncgen

The `ncgen` tool generates a netCDF file or the C or FORTRAN programs required to create the netCDF file. If no options are specified in invoking `ncgen`, the program merely checks the syntax of the CDL input, producing error messages for any violations of CDL syntax. Other options can be used to create a netCDF file or to generate a program in C or FORTRAN that calls the routines required to create the netCDF file.

UNIX syntax for invoking `ncgen`:

```
ncgen [-b] [-o output-file] [-c] [-f] [-n] [input-file]
```

where:

'`-b`' Create a (binary) netCDF file. If the '`-o`' option is absent, a default file name will be constructed from the netCDF name (specified after the `netcdf` keyword in the input) by appending the '`.nc`' extension. If a file already exists with the specified name it will be overwritten.

'`-o outputfile`'
Name for the netCDF file created. If this option is specified, it implies the '`-b`' option. (This option is necessary because netCDF files are direct-access files created with seek calls, and hence cannot be written to standard output.)

'`-c`' Generate C source code that will create a netCDF file matching the netCDF specification. The C source code is written to standard output.

'`-f`' Generate FORTRAN source code that will create a netCDF file matching the netCDF specification. The FORTRAN source code is written to standard output.

'`-n`' Like the '`-b`' option, except creates a netCDF file with the obsolete '`.cdf`' extension instead of the '`.nc`' extension, in the absence of an output filename specified by the '`-O`' option. This option is only supported for backward compatibility.

Examples

Check the syntax of the CDL file '`foo.cdl`':

```
ncgen foo.cdl
```

From the CDL file '`foo.cdl`', generate an equivalent binary netCDF file named '`flu.nc`':

```
ncgen -o flu.nc foo.cdl
```

From the CDL file '`foo.cdl`', generate a C program containing the netCDF function invocations necessary to create an equivalent binary netCDF file named '`flu.nc`':

```
ncgen -c -o flu.nc foo.cdl
```

10.5 ncdump

The `ncdump` tool generates an ASCII representation of a netCDF file on standard output, either with or without an ASCII representation of the variable data in the file. The ASCII representation used is the CDL notation that `ncgen` accepts as input. Thus `ncdump` and `ncgen` can be used as inverses to transform data representation between binary and ASCII representations.

`ncdump` may also be used as a simple browser for netCDF data files, to display the dimension names and sizes; variable names, types, and shapes; attribute names and values; and optionally, the values of data for all variables or selected variables in a netCDF file.

`ncdump` defines a default format used for each type of netCDF variable data, but this can be overridden if a `C_format` attribute is defined for a netCDF variable. In this case, `ncdump` will use the `C_format` attribute to format values for that variable. For example, if floating-point data for the netCDF variable `Z` is known to be accurate to only three significant digits, it would be appropriate to use the variable attribute

```
Z:C_format = "%.3g"
```

UNIX syntax for invoking `ncdump`:

```
ncdump  [ -c | -h]  [-v var1,...]  [-b lang]  [-f lang]
[-l len]  [ -d fdig[,ddig]] [ -n name]  [input-file]
```

where:

'-c' Show the values of *coordinate* variables (variables that are also dimensions) as well as the declarations of all dimensions, variables, and attribute values. Data values of non-coordinate variables are not included in the output. This is often the most suitable option to use for a brief look at the structure and contents of a netCDF file.

'-h' Show only the *header* information in the output, that is, output only the declarations for the netCDF dimensions, variables, and attributes of the input file, but no data values for any variables. The output is identical to using the '-c' option except that the values of coordinate variables are not included. (At most one of '-c' or '-h' options may be present.)

'-v var1,...'

The output will include data values for the specified variables, in addition to the declarations of all dimensions, variables, and attributes. One or more variables must be specified by name in the comma-delimited list following this option. The list must be a single argument to the command, hence cannot contain blanks or other white space characters. The named variables must be valid netCDF variables in the input-file. The default, without this option and in the absence of the '-c' or '-h' options, is to include data values for *all* variables in the output.

'-b lang' A brief annotation in the form of a CDL comment (text beginning with the characters '//') will be included in the data section of the output for each 'row' of data, to help identify data values for multidimensional variables. If *lang* begins with 'C' or 'c', then C language conventions will be used (zero-based indices, last dimension varying fastest). If *lang* begins with 'F' or 'f', then Fortran language conventions will be used (one-based indices, first dimension varying fastest). In either case, the data will be presented in the same order; only the annotations will differ. This option is useful for browsing through large volumes of multidimensional data.

'-f lang' Full annotations in the form of trailing CDL comments (text beginning with the characters '//') for every data value (except individual characters in character arrays) will be included in the data section. If *lang* begins with 'C' or 'c', then C language conventions will be used (zero-based indices, last dimension varying fastest). If *lang* begins with 'F' or 'f', then Fortran language conventions will be used (one-based indices, first dimension varying fastest). In either case, the data will be presented in the same order; only the annotations will differ. This option may be useful for piping data into other filters, since each data value appears on a separate line, fully identified.

'-l len' Changes the default maximum line length (80) used in formatting lists of non-character data values.

'-d float_digits[,double_digits]'

Specifies default number of significant digits to use in displaying floating-point or double precision data values for variables that don't have a 'C_format' attribute. Floating-point data will be displayed with *float_digits* significant digits. If *double_digits* is also specoified, double-precision values will be displayed with that many significant digits. If a variable has a 'C_format' attribute, that overrides any specified floating-point default. In the absence of any '-d' specifications, floating-point and double-precision data are displayed with 7 and 15 significant digits respectively. CDL files can be made smaller if less precision is required. If both floating-point and double-presision precisions are specified, the two values must appear separated by a comma (no blanks) as a single argument to the command.

'-n name' CDL requires a name for a netCDF data set, for use by 'ncgen -b' in generating a default netCDF file name. By default, ncdump constructs this name from the last component of the pathname of the input netCDF file by stripping off any extension it has. Use the '-n' option to specify a different name. Although the output file name used by 'ncgen -b' can be specified, it may be wise to have ncdump change the default name to avoid inadvertantly overwriting a valuable netCDF file when using ncdump, editing the resulting CDL file, and using 'ncgen -b' to generate a new netCDF file from the edited CDL file.

Examples

Look at the structure of the data in the netCDF file 'foo.nc':

```
ncdump -c foo.nc
```

Produce an annotated CDL version of the structure and data in the netCDF file 'foo.nc', using C-style indexing for the annotations:

```
ncdump -b c foo.nc > foo.cdl
```

Output data for only the variables uwind and vwind from the netCDF file 'foo.nc', and show the floating-point data with only three significant digits of precision:

```
ncdump -v uwind,vwind -d 3 foo.nc
```

Produce a fully-annotated (one data value per line) listing of the data for the variable `omega`, using Fortran conventions for indices, and changing the netCDF dataset name in the resulting CDL file to `omega`:

```
ncdump -v omega -f fortran -n omega foo.nc > Z.cdl
```

11 Answers to Some Frequently Asked Questions

Here we collect, in one place, answers to some of the most frequently asked questions about netCDF availability and support. A more comprehensive and up-to-date list of answers to frequently asked questions about netCDF is maintained in a file available for anonymous FTP access from:

```
host:  unidata.ucar.edu
file:  pub/netcdf/FAQ
```

What is netCDF?

NetCDF (network Common Data Form) is an interface for scientific data access and a freely-distributed software library that provides an implementation of the interface. It was developed by Glenn Davis, Russ Rew, and Steve Emmerson at the Unidata Program Center in Boulder, Colorado. The netCDF library also defines a machine-independent format for representing scientific data. Together, the interface, library, and format support the creation, access, and sharing of scientific data.

NetCDF data is:

- Self-Describing. A netCDF file includes information about the data it contains.
- Network-transparent. A netCDF file is represented in a form that can be accessed by computers with different ways of storing integers, characters, and floating-point numbers.
- Direct-access. A small subset of a large dataset may be accessed efficiently, without first reading through all the preceding data.
- Appendable. Data can be appended to a netCDF dataset along one dimension without copying the dataset or redefining its structure. The structure of a netCDF dataset can be changed, though this sometimes causes the dataset to be copied.
- Sharable. One writer and multiple readers may simultaneously access the same netCDF file.

How do I get the netCDF software package?

Via anonymous FTP from

```
host:       unidata.ucar.edu
```

```
file:      pub/netcdf/netcdf.tar.Z
```

Make sure the file is transmitted in BINARY mode. This is version 2.3, last updated in February 1993, or later.

What does netCDF run on?

Version 2.3 of netCDF has been tested on the following platforms:

- SPARCstation, SunOS 4.1.3
- DECstation or VAXstation, Ultrix 4.3
- VAXstation, VMS
- IBM RS-6000, AIX 3.x
- IBM PS/2, MSDOS 5.0
- IBM PS/2, OS/2
- CRAY YMP, UNICOS
- SGI, IRIX
- HP, HPUX

The experience of outside developers indicates that netCDF is relatively easy to port to any system that has a C compiler and an XDR library (eXternal Data Representation, used for NFS and usually supplied by vendors).

What is the connection between netCDF and CDF?

CDF was developed at the NASA Space Science Data Center at Goddard, and is freely available. It was originally a VMS FORTRAN interface for scientific data access. Unidata reimplemented the library from scratch to use XDR for a machine-independent representation, designed the CDL text representation for netCDF data, wrote a User's Guide and made other additions including aggregate data access, single-file implementation, named dimensions, and variable-specific attributes.

NetCDF and CDF have evolved independently. CDF now supports many of the same features as netCDF (aggregate data access, XDR representation, single-file representation, variable-specific attributes), but some differences remain (netCDF doesn't support native-mode representation, CDF doesn't support named dimensions).

What is the connection between netCDF and HDF?

The National Center for Supercomputing Applications (NCSA) developed the HDF software and makes it freely available. HDF is an extensible data format for self-describing files that was developed independently of netCDF. Applications and utilities based on HDF are available that support raster-image manipulation and display and browsing through multidimensional scientific data. The HDF software includes a package of routines for accessing each HDF data type, as well as a lower-level interface for building packages to support new types. HDF supports both C and Fortran interfaces, and it has been successfully ported to a wide variety of machine architectures and operating systems. HDF emphasizes a single common format for data, on which many interfaces can be built.

NCSA has implemented software that provides a netCDF interface to HDF. With this software, it is possible to use the netCDF calling interface to place data into an HDF file. The netCDF calling interface has not changed and netCDF files stored in XDR format are readable, so existing programs and data will still be usable (although programs will need to be relinked to the new library). There is currently no support for the mixing of HDF and netCDF structures. For example, a raster image can exist in the same file as a netCDF object, but you have to use the Raster Image interface to read the image and the netCDF interface to read the netCDF object. The other HDF interfaces are currently being modified to allow multi-file access, closer integration with the netCDF interface will probably be delayed until the end of that project.

Eventually, it may be possible to integrate netCDF objects with the rest of the HDF tool suite. Such an integration will then allow tools written for netCDF and tools written for HDF to both interact intelligently with the new data files.

What are the files in pub/netcdf/ on unidata.ucar.edu?

'`README`' general information about netCDF

'`FAQ`' this file of frequently asked questions

'`utilities.txt`'
: a list of software packages currently available or under development for manipulating and displaying netCDF data

'guide.ps.Z'
: a compressed PostScript file of the NetCDF User's Guide. This is included in the netcdf.tar.Z distribution, so you don't need both.

'ncprogs.ps'
: a draft PostScript document describing an initial set of netCDF operator and utility programs under development

'ncprogs.txt'
: an ASCII version of ncprogs.ps

'conventions.info'
: a draft document of some proposed netCDF conventions

'cdl/' a directory containing some examples of CDL files (an ASCII representation for netCDF files).

'msdos/' a directory containing executables and binaries for netCDF under MSDOS 5.0. These can also be built from the sources in netcdf.tar.Z, if you have the necessary Microsoft compilers.

'mac/' a directory containing notes and Macintosh MPW makefiles for porting netCDF to an Apple Macintosh. These were contributed by Chuck Denham, U.S. Geological Survey.

Why do I get XDR error messages when trying to write data?

Probably due to a write error, perhaps because of exceeding disk quotas, a full device, or permission problems. The netCDF library is usually built on a vendor-supplied XDR library layer. Sometimes errors occur in the XDR layer rather than the netCDF layer, and in this case the error messages from the XDR layer can be cryptic. For example, if the user is writing data and exceeds a disk space quota, this might be detected first in the XDR layer. Although we try to catch and elaborate on errors that occur in the XDR layer, sometimes there is not enough information passed up to calling routines from the XDR layer to make construction of a meaningful error message possible.

What's the best way to represent data using netCDF?

There are many ways to represent the same information in any general-purpose data model. Choices left up to the user in the case of netCDF include which information to represent as variables or as variable attributes; what names to choose for variables, dimensions, and attributes; what order to use for the dimensions of multidimensional variables; what variables to include in the same netCDF file; and how to use variable attributes to capture the structure and meaning of data. We provide some guidelines in the NetCDF User's Guide (See ⟨undefined⟩ [Attributes and Variables], page ⟨undefined⟩), but we've found that a little experience helps. Occasionally we have decided it was useful to change the structure of netCDF files after experience with how the data is used.

Are there plans to add facilities for data compression to netCDF?

We have no plans to add data compression to netCDF (although we do plan to eventually add a form of transparent data packing on write and unpacking on read whenever the reserved attributes `_Nbits`, `_Scale`, and `_Offset` are defined).

Hyperslab access and direct access to individual array values conflict with most simple compression schemes. With netCDF, the elements of an array variable can be filled in any order or as cross-sections in any direction. NetCDF permits writing elements in one order and reading them later in different orders.

Some compression methods require that all the data to be compressed are known before starting the compression. Techniques like run-length encoding or anything that depends on exploiting similarities in nearby values can't be used if nearby values aren't all known at the time some of the data are to be written.

An alternative that can be implemented above the netCDF library is to adopt a convention for compressed data that uses a "compression" attribute to encode the method of compression, e.g.

```
x:compression = "rle" ;
```

for run-length encoding of the data in a variable `x`. Then when you write the data, compress them into a bland array of bytes and write all the bytes. Note that it would be difficult to define the size of such a variable in advance, since its compressed size depends on its values. You would also have to give up on hyperslab access for such variables, but instead read the compressed array in all at once and uncompress it before using it.

What utilities are available for netCDF?

The only utilities available in the current netCDF distribution are ncdump and ncgen, for converting netCDF files to an ASCII human-readable form and for converting from the ASCII human-readable form back to a binary netCDF file or a C or FORTRAN program for generating the netCDF file. In addition, the first release of the netcdf operators package, containing a few general-purpose netCDF operators, is available from `unidata.ucar.edu` in the file '`ncopers.tar.Z`'. See the description of the file '`utilities.txt`' in the answer to the question above on important files in '`/pub/netcdf`' on `unidata.ucar.edu` for more information.

Is there a mailing list for netCDF discussions and questions?

Yes - there are two. The mailing list `netcdfgroup@unidata.ucar.edu` has over two hundred readers and gateways; you can subscribe or unsubscribe to the mailing list by sending mail to `netcdfgroup-adm@unidata.ucar.edu`.

If you would prefer to get only a single daily digest of the postings to the `netcdfgroup` mailing list, subscribe instead to the `netcdfdigest` mailing list by sending a request to `netcdfdigest-adm@unidata.ucar.`

All the postings to the `netcdfgroup` mailing list are archived in the file '`mail-archives/netcdfgroup`' available via anonymous FTP from `unidata.ucar.edu`.

How do I make a bug report?

If you find a bug, please send a description to `support@unidata.ucar.edu`.

This is also a better address to use for questions or discussions about netCDF that you think are not appropriate for the entire `netcdfgroup` mailing list.

How widely is netCDF used?

We don't really know, but the netcdfgroup and netcdfdigest mailing lists have 260 addresses, some of which are aliases to more addresses, in fourteen countries. Users in several groups have adopted netCDF as a standard way to represent at least some kinds of scientific datasets.

Is there any WAIS or gopher access to netCDF information?

Yes, '`netcdf-group.src`' is a WAIS source that provides a full-text search of the netcdfgroup mailing list archive and is registered with think.com for general use.

Another WAIS source, '`unidata-support-netcdf.src`', provides a full-text search of the support questions and answers about netCDF provided by Unidata support staff.

Both of these WAIS servers and access to other information about Unidata can be accessed most easily through the Unidata gopher server at gopher.unidata.ucar.edu.

Appendix A Summary of C Interface

```
int nccreate (const char* path, int cmode);
int ncopen (const char* path, int mode);
int ncredef (int ncid);
int ncendef (int ncid);
int ncclose (int ncid);
int ncinquire (int ncid, int* ndims, int* nvars, int* natts, int* recdim);
int ncsync (int ncid);
int ncabort (int ncid);
int ncdimdef (int ncid, const char* name, long length);
int ncdimid (int ncid, const char* name);
int ncdiminq (int ncid, int dimid, char* name, long* length);
int ncdimrename (int ncid, int dimid, const char* name);
int ncvardef (int ncid, const char* name, nc_type datatype, int ndims,
              const int dimids[]);
int ncvarid (int ncid, const char* name);
int ncvarinq (int ncid, int varid, char* name, nc_type* datatype, int* ndims,
              int dimids[], int* natts);
int ncvarput1 (int ncid, int varid, const long coords[], const void* value);
int ncvarget1 (int ncid, int varid, const long coords[], void* value);
int ncvarput (int ncid, int varid, const long start[], const long count[],
              const void* value);
int ncvarget (int ncid, int varid, const long start[], const long count[],
              void* value);
int ncvarputg (int ncid, int varid, const long start[], const long count[],
              const long stride[], const long imap[], const void* value);
int ncvargetg (int ncid, int varid, const long start[], const long count[],
              const long stride[], const long imap[], void* value);
int ncvarrename (int ncid, int varid, const char* name);
int ncattput (int ncid, int varid, const char* name, nc_type datatype, int len,
              const void* value);
int ncattinq (int ncid, int varid, const char* name, nc_type* datatype,
              int* len);
int ncattget (int ncid, int varid, const char* name, void* value);
int ncattcopy (int incdf, int invar, const char* name, int outcdf, int outvar);
int ncattname (int ncid, int varid, int attnum, char* name);
int ncattrename (int ncid, int varid, const char* name, const char* newname);
int ncattdel (int ncid, int varid, const char* name);
int nctypelen (nc_type datatype);
int ncsetfill (int ncid, int fillmode);
int ncrecput(int ncid, long recnum, void* const datap[]);
int ncrecget(int ncid, long recnum, void* datap[]);
int ncrecinq(int ncid, int *nrecvars, int recvarids[], long recsizes[]);
```

Appendix B Summary of FORTRAN Interface

Input parameters are in upper case, output parameters are in lower case. The FORTRAN types of all the parameters are listed below the subroutine and function declarations.

```
INTEGER FUNCTION NCCRE (PATHNAME,CLOBMODE, rcode)
INTEGER FUNCTION NCOPN (PATHNAME,RWMODE, rcode)
SUBROUTINE NCREDF (NCID, rcode)
SUBROUTINE NCENDF (NCID, rcode)
SUBROUTINE NCCLOS (NCID, rcode)
SUBROUTINE NCINQ (NCID, ndims,nvars,natts,recdim,rcode)
SUBROUTINE NCSNC (NCID, rcode)
SUBROUTINE NCABOR (NCID, rcode)
INTEGER FUNCTION NCDDEF (NCID,DIMNAME,SIZE, rcode)
INTEGER FUNCTION NCDID (NCID,DIMNAME, rcode)
SUBROUTINE NCDINQ (NCID,DIMID, dimname,size,rcode)
SUBROUTINE NCDREN (NCID,DIMID,DIMNAME, rcode)
INTEGER FUNCTION NCVDEF (NCID,VARNAME,DATATYPE,NVDIMS,VDIMS, rcode)
INTEGER FUNCTION NCVID (NCID,VARNAME, rcode)
SUBROUTINE NCVINQ (NCID,VARID, varname,datatype,nvdims,vdims,nvatts,rcode)
SUBROUTINE NCVPT1 (NCID,VARID,INDICES,VALUE, rcode)
SUBROUTINE NCVP1C (NCID,VARID,INDICES, CHVAL, rcode)
SUBROUTINE NCVGT1 (NCID,VARID,INDICES, value, rcode)
SUBROUTINE NCVG1C (NCID,VARID,INDICES, chval, rcode)
SUBROUTINE NCVPT (NCID,VARID,START,COUNTS,VALUE, rcode)
SUBROUTINE NCVPTC (NCID,VARID,START,COUNTS,STRING,LENSTR, rcode)
SUBROUTINE NCVPTG (NCID,VARID,START,COUNTS,STRIDE,IMAP,VALUE, rcode)
SUBROUTINE NCVPGC (NCID,VARID,START,COUNTS,STRIDE,IMAP,STRING,rcode)
SUBROUTINE NCVGT (NCID,VARID,START,COUNTS, value,rcode)
SUBROUTINE NCVGTC (NCID,VARID,START,COUNTS, string,LENSTR,rcode)
SUBROUTINE NCVGTG (NCID,VARID,START,COUNTS,STRIDE,IMAP,value,rcode)
SUBROUTINE NCVGGC (NCID,VARID,START,COUNTS,STRIDE,IMAP,string,rcode)
SUBROUTINE NCVREN (NCID,VARID,VARNAME, rcode)
SUBROUTINE NCAPT (NCID,VARID,ATTNAME,DATATYPE,ATTLEN,VALUE, rcode)
SUBROUTINE NCAPTC (NCID,VARID,ATTNAME,DATATYPE,LENSTR,string, rcode)
SUBROUTINE NCAINQ (NCID,VARID,ATTNAME, datatype,attlen,rcode)
SUBROUTINE NCAGT (NCID,VARID,ATTNAME, value,rcode)
SUBROUTINE NCAGTC (NCID,VARID,ATTNAME, string,LENSTR,rcode)
SUBROUTINE NCACPY (INNCID,INVARID,ATTNAME,OUTNCID,OUTVARID, rcode)
SUBROUTINE NCANAM (NCID,VARID,ATTNUM, attname,rcode)
SUBROUTINE NCAREN (NCID,VARID,ATTNAME,NEWNAME, rcode)
SUBROUTINE NCADEL (NCID,VARID,ATTNAME, rcode)
INTEGER FUNCTION NCTLEN (DATATYPE, rcode)
SUBROUTINE NCPOPT (NCOPTS)
SUBROUTINE NCGOPT (ncopts)
INTEGER FUNCTION NCSFIL (NCID,FILLMODE, rcode)

CHARACTER*(*) PATHNAME  ! absolute or relative name of netCDF file
```

```
INTEGER CLOBMODE        ! either NCCLOB or NCNOCLOB
INTEGER RWMODE          ! either NCWRITE or NCNOWRIT
INTEGER RCODE           ! returned error code, 0 if no errors
INTEGER NCID            ! netCDF ID, returned by NCCRE or NCOPN
INTEGER NDIMS           ! number of dimensions in netCDF
INTEGER NVARS           ! number of variables in netCDF
INTEGER NATTS           ! number of global attributes in netCDF
INTEGER RECDIM          ! dimension ID of unlimited dimension
CHARACTER*(*) DIMNAME   ! name for dimension
INTEGER SIZE            ! size of dimension
INTEGER DIMID           ! dimension ID from NCDDEF or NCDID
CHARACTER*(*) VARNAME   ! name for variable
INTEGER DATATYPE        ! data type code, one of NCBYTE, ..., NCDOUBLE
INTEGER NVDIMS          ! number of dimensions in a variable
INTEGER VDIMS(NDIMS)    ! dimension IDs for a variable, giving its shape
INTEGER VARID           ! variable ID from NCVDEF or NCVID, or NCGLOBAL
INTEGER NVATTS          ! number of attributes assigned to a variable
INTEGER INDICES(NDIMS)  ! coordinates of a single element of a variable
CHARACTER CHVAL         ! character value of variable or attribute
CHARACTER*(*) STRING    ! character array value of variable or attribute
INTEGER LENSTR          ! length of character array value
DOUBLE VALUE            ! double precision value of variable or attribute
REAL VALUE              ! real value of variable or attribute
INTEGER VALUE           ! integer value of variable or attribute
INTEGER START(NVDIMS)   ! corner of hyperslab of values of a variable
INTEGER COUNTS(NVDIMS)  ! edge lengths of hyperslab of values
INTEGER STRIDE(NVDIMS)  ! netCDF variable dimensional strides
INTEGER IMAP(NVDIMS)    ! index mapping vector
CHARACTER*(*) ATTNAME   ! attribute name
INTEGER ATTLEN          ! number of elements in an attribute vector
INTEGER INNCID          ! input netCDF ID
INTEGER INVARID         ! input variable ID
INTEGER OUTNCID         ! output netCDF ID
INTEGER OUTVARID        ! output variable ID
INTEGER ATTNUM          ! attribute number
CHARACTER*(*) NEWNAME   ! new attribute name
INTEGER FILLMODE        ! NCNOFILL or NCFILL, for setting fill mode
```

Appendix C Units

The Unidata Program Center has developed a units library to convert between formatted and binary forms of units specifications and perform unit algebra on the binary form. Though the units library is self-contained and there is no dependency between it and the netCDF library, it is nevertheless useful in writing generic netCDF programs and we suggest you obtain it. A compressed tar file for the library is available from the file `pub/sdm/udunits<version>.tar.Z` in the anonymous FTP directory of `unidata.ucar.edu`.

The following are examples of units strings that can be interpreted by the `utScan()` function of the Unidata units library:

```
10 kilogram.meters/seconds2
10 kg-m/sec2
10 kg m/s^2
10 kilogram meter second-2
(PI radian)2
degF
100rpm
geopotential meters
33 feet water
milliseconds since 1992-12-31 12:34:0.1 -7:00
```

A unit is specified as an arbitrary product of constants and unit-names raised to arbitrary integral powers. Division is indicated by a slash '`/`'. Multiplication is indicated by whitespace, a period '`.`', or a hyphen '`-`'. Exponentiation is indicated by an integer suffix or by the exponentiation operators '`^`' and '`**`'. Parentheses may be used for grouping and disambiguation. The timestamp in the last example is handled as a special case.

Arbitrary Galilean transformations (i.e. $y = ax + b$) are allowed. In particular, temperature conversions are correctly handled. The specification:

```
degF @ 32
```

indicates a Fahrenheit scale with the origin shifted to thirty-two degrees Fahrenheit (i.e. to zero Celsius). Thus, the Celsius scale is equivalent to the following unit:

```
1.8 degF @ 32
```

Note that the origin-shift operation takes precedence over multiplication. In order of increasing precedence, the operations are division, multiplication, origin-shift, and exponentiation.

`utScan()` understands all the SI prefixes (e.g. "mega" and "milli") plus their abbreviations (e.g. "M" and "m")

The function utPrint() always encodes a unit specification one way. To reduce misunderstandings, it is recommended that this encoding style be used as the default. In general, a unit is encoded in terms of basic units, factors, and exponents. Basic units are separated by spaces, and any exponent directly appends its associated unit. The above examples would be encoded as follows:

```
10 kilogram meter second-2
9.8696044 radian2
0.555556 kelvin @ 255.372
10.471976 radian second-1
9.80665 meter2 second-2
98636.5 kilogram meter-1 second-2
0.001 seconds since 1992-12-31 19:34:0.1000 UTC
```

(Note that the Fahrenheit unit is encoded as a deviation, in fractional kelvins, from an origin at 255.372 kelvin, and that the time in the last example has been referenced to UTC.)

The database for the units library is a formatted file containing unit definitions and is used to initialize this package. It is the first place to look to discover the set of valid names and symbols.

The format for the units-file is documented internally and the file may be modified by the user as necessary. In particular, additional units and constants may be easily added (including variant spellings of existing units or constants).

`utScan()` is case-sensitive. If this causes difficulties, you might try making appropriate additional entries to the units-file.

Some unit abbreviations in the default units-file might seem counter-intuitive. In particular, note the following:

```
For       Use               Not     Which Instead Means

Celsius   `Celsius'         `C'     coulomb
gram      `gram'            `g'     <standard free fall>
```

```
gallon    ‘gallon’         ‘gal’   <acceleration>
radian    ‘radian’         ‘rad’   <absorbed dose>
Newton    ‘newton’ or ‘N’   ‘nt’    nit (unit of photometry)
```

For additional information on this units library, please consult the manual pages that come with the distribution.

Function and Variable Index

(Index is nonexistent)

General Index

Table of Contents