NCACPY: FORTRAN Interface 106
8.5 Get Name of Attribute from Its Number 108
ncattname: C Interface 108
NCANAM: FORTRAN Interface 109
8.6 Rename an Attribute 110
ncattrename: C Interface 110
NCAREN: FORTRAN Interface 111
8.7 Delete an Attribute 112
ncattdel: C Interface 112
NCADEL: FORTRAN Interface 113

9 Miscellaneous netCDF Operations 115
9.1 Get Number of Bytes for a Data Type 116
nctypelen: C Interface 116
NCTLEN: FORTRAN Interface 117

10 Higher-Level netCDF Operations 119
10.1 ncgen 120
CDL Syntax 120
CDL Data Types 122
CDL Notation for Data Constants 123
10.2 ncdump 126

11 How to Obtain netCDF Software 127

12 Summary of C Interface 129

13 Summary of FORTRAN Interface 131

Function and Variable Index 133

General Index 135

7 Variables 65

7.1 Create a Variable 66
ncvardef: C Interface 66
NCVDEF: FORTRAN Interface 67
7.2 Get a Variable ID from Its Name 69
ncvarid: C Interface 69
NCVID: FORTRAN Interface 69
7.3 Get Information about a Variable from Its ID 71
ncvarinq: C Interface 71
NCVINQ: FORTRAN Interface 72
7.4 Write a Single Data Value 74
ncvarput1: C Interface 74
NCVPT1: FORTRAN Interface 75
7.5 Write a Hyperslab of Values 77
ncvarput: C Interface 77
NCVPT: FORTRAN Interface 79
7.6 Read a Single Data Value 81
ncvarget1: C Interface 81
NCVGT1: FORTRAN Interface 82
7.7 Read a Hyperslab of Values 84
ncvarget: C Interface 84
NCVGT, NCVGTC: FORTRAN Interface 86
7.8 Reading and Writing Character String Values 88
C Interface 88
FORTRAN Interface 89
7.9 Missing Values 91
7.10 Rename a Variable 92
ncvarrename: C Interface 92
NCVREN: FORTRAN Interface 93

8 Attributes 95

8.1 Create an Attribute 96
ncattput: C Interface 96
NCAPT, NCAPTC: FORTRAN Interface 97
8.2 Get Information about an Attribute 99
ncattinq: C Interface 99
NCAINQ: FORTRAN Interface 100
8.3 Get Attribute's Values 102
ncattget: C Interface 102
NCAGT, NCAGTC: FORTRAN Interface 103
8.4 Copy Attribute from One netCDF to Another 105
ncattcopy: C Interface 105

5 netCDF Operations 37

5.1 netCDF Library Interface Descriptions 37
5.2 Create a netCDF file 39
nccreate: C Interface 39
NCCRE: FORTRAN Interface 40
5.3 Open a netCDF File for Access 41
ncopen: C Interface 41
NCOPN: FORTRAN Interface 41
5.4 Put Open netCDF File into Define Mode 43
ncredef: C Interface 43
NCREDF: FORTRAN Interface 43
5.5 Leave Define Mode 45
ncendef: C Interface 45
NCENDF: FORTRAN Interface 45
5.6 Close an Open netCDF File 47
ncclose: C Interface 47
NCCLOS: FORTRAN Interface 48
5.7 Inquire about an Open netCDF File 49
ncinquire: C Interface 49
NCINQ: FORTRAN Interface 50
5.8 Synchronize an Open netCDF File to Disk 51
ncsync: C Interface 51
NCSNC: FORTRAN Interface 52
5.9 Back Out of Recent Definitions 53
ncabort: C Interface 53
NCABOR: FORTRAN Interface 54

6 Dimensions 55

6.1 Create a Dimension 56
ncdimdef: C Interface 56
NCDDEF: FORTRAN Interface 57
6.2 Get a Dimension ID from Its Name 58
ncdimid: C Interface 58
NCDID: FORTRAN Interface 59
6.3 Inquire about a Dimension 60
ncdiminq: C Interface 60
NCDINQ: FORTRAN Interface 61
6.4 Rename a Dimension 63
ncdimrename: C Interface 63
NCDREN: FORTRAN Interface 64

Table of Contents

Acknowledgments **1**

Foreword **3**

Summary **5**

1 Introduction **7**

1.1 The netCDF Interface 7
1.2 The netCDF is Not a Database Management System 7
1.3 What about Performance? 8
1.4 Is netCDF a Good Archive Format? 9
1.5 Background and Evolution of the netCDF Interface 9
1.6 Future Plans for netCDF 11
References 11

2 Components of a netCDF File **13**

2.1 Dimensions 14
2.1.1 Using Dimensions to Specify Variable Shapes 15
2.1.2 Using Dimensions to Relate Variables 15
2.1.3 Using Dimensions to Define Coordinate Systems 16
2.2 Variables 16
2.3 Attributes 18
2.3.1 Attribute Conventions 19
2.3.2 Differences between Attributes and Variables 21

3 Data **23**

3.1 netCDF Data Types 23
3.2 Data Access 24
3.3 Data Structures 26

4 Use of the netCDF Library **29**

4.1 Creating a netCDF File 29
4.2 Reading a netCDF File with Known Names 31
4.3 Reading a netCDF File with Unknown Names 32
4.4 Adding New Dimensions, Variables, Attributes 34
4.5 Error Handling 35

reading data 81, 84, 88
record dimension 14, 55, 56, 60, 65
record variables 14, 17, 26
record-oriented access 26
recording data history 20
records 26, 55, 60
relating variables 15
relational database systems 7
removing attributes 112
renaming attributes 34, 43, 110
renaming dimensions 34, 43, 63
renaming variables 34, 43, 92
reporting bugs 127
representation on disk 26
restoring old definitions 35

S

scalar variables 17
scale_factor attribute 20
scaling data 20
SeaSpace CDF 10
self-describing 5
shape of a netCDF variable 16
shared access 51
short CDL type 17
short constant 124
short type 23, 123, 124
sparse matrices 26
specifying variable shapes 15
string length, actual 89
string length, declared 89
string-valued attributes 18
subscript order 24, 25
suppressing error messages 35
synchronize a netCDF file 51

T

title attribute 20
trees 26
type lengths 116
typical netCDF calls 29

U

Unidata CDF Workshops 10
units attribute 18, 19, 21
unlimited dimension 14, 15, 17, 55, 56, 60, 65
unlimited dimension ID 49, 50
unsigned 23

V

valid_max attribute 20
valid_min attribute 20
valid_range attribute 19
variable 106
variable attributes 16, 18, 19, 71, 72, 96, 99, 122
variable data 31, 74, 75, 77, 79, 81, 82, 84, 86, 97
variable data types 17
variable data values 16
variable dimensions 71, 72, 74, 75, 77, 79, 81, 82, 84, 86, 97
variable ID 65, 66, 69
variable IDs 31
variable indices 74, 75, 77, 79, 81, 82, 84, 86, 97
variable inquire 71
variable name 16, 19, 65, 66, 69, 71, 72, 92, 93
variable shape 16, 65, 66, 71, 88
variable size 17
variable type 65, 66, 71, 72
variable values 21, 65
variable-length strings 88, 89
variables 15, 16, 18, 21, 30, 31, 32, 33, 65, 105

W

writing character string data 88
writing data 74, 77, 88

X

XDR 8, 10, 23, 26

index variables 26
inquire about a netCDF file 49
inquire functions 32
interface descriptions 37

K

known names 31

L

language interfaces 26
languages supported 7
linked lists 26
long CDL type 17
long constant 124
long type 23, 123, 124
long_name attribute 19

M

mailing list 127
MAX_NC_DIMS 55
maximum dimensions 66
maximum name length 71, 72, 108, 109
maximum number of dimensions 55
maximum number of records 60
maximum variable dimensions 66, 67, 71, 72
metadata 18, 21
missing values 20, 91
missing_value attribute 20

N

NASA CDF 9
NC_BYTE type specifier 17
NC_CHAR type specifier 17
NC_DOUBLE type specifier 17
NC_FLOAT type specifier 17
NC_LONG type specifier 17
NC_SHORT type specifier 17
netCDF 7
netCDF attributes 13
netCDF components 13
netCDF data types 23, 116
netCDF dimensions 13
netCDF disk representation 88
netCDF file creation 29, 39
netCDF file extension 120
netCDF file name 37
netCDF file size 88, 89
netCDF handle 37
netCDF ID 37, 47
netCDF implementation 55
netCDF library use 29
netCDF names 14
netCDF operations 37
netCDF variables 13
netCDF, development of 9
netcdfgroup 127
network Common Data Form Language (CDL) 13
network-transparent 5
New Mexico Tech. CDF 10
NSSDC CDF 9
null byte 88, 89
number of dimensions 49
number of global attributes 49
number of records written 60
number of variables 49
numeric values 89

O

obtaining netCDF software 127
opening a netCDF file 31, 41
operating systems supported 7
order of data 26
order of dimensions 26

P

performance 8, 26, 30, 31, 51
pointers 26
portability 7
ports of netCDF 127
primitive netCDF types 23
putting character string data 88
putting variable data 74, 77, 84

R

reading a netCDF file 31
reading character string data 88

data mode 30, 34, 45, 51, 53, 95
data order 24, 25, 26
data packing 20
data resolution 20
data section 122
data sizes 116
data structures 26
data types 16, 17, 66, 67, 71, 72, 75, 79, 82, 86, 96, 99, 100, 117
database management systems 7, 18
default error handling 35
define mode 30, 34, 43, 45, 51, 53, 55, 65, 66, 95
defining attributes 96
defining coordinate systems 16
deleting a netCDF file 53
deleting attributes 34, 43, 112
differences between attributes and variables 21
dimension ID 55
dimension IDs 31, 33, 56, 58, 60, 66
dimension inquire 60
dimension name 55
dimension names 14, 56, 58, 60, 61, 63
dimension size 14, 55, 56, 60
dimensions 14, 18, 30, 31, 32, 33, 55
double CDL type 17
double constant 125
double type 23, 123, 124, 125

E

efficiency 8, 26, 30, 31, 51
error conditions 37
error handling 35
error messages 35
error options 35
error returns 35
example conventions 37
examples 37
external data representation (XDR) 8

F

fatal errors 35
fill values 20, 91
fixed-length character strings 88
fixed-size strings 88, 89
float CDL type 17
float constant 124
float data type 123
float type 23, 124
FORTRAN interface 5
FORTRAN, generation of 120
FORTRAN_format attribute 20
FTP access 127
function prototypes 37
future changes planned 7, 23, 123
Future Plans 11

G

generating code 120
generating netCDF files 120
generic applications 18, 19, 20, 31, 32, 55, 108
generic filters 20
getting attribute name 108
getting attribute values 102
getting character string data 88
getting dimension ID 58
getting dimension name 60
getting dimension size 60
getting variable data 81
getting variable name 71
getting variable shape 71
getting variable type 71
global attributes ... 18, 19, 49, 95, 96, 99, 100, 103, 122

H

higher-level netCDF operations 119
history attribute 20
history of the netCDF 9
hyperslab access 31, 77, 84, 88
hyperslab access example 24
hyperslab corner 24, 25, 88
hyperslab edge lengths 24, 25, 88
hyperslabs 24

I

IEEE floating-point 8, 23
index order 24, 25

General Index

A

abnormal termination 30
aborting define mode 35
aborting definitions 53
abstract data type 7
add_offset attribute 20
adding attributes 34, 43
adding dimensions 34, 43
adding variables 34, 43
archive formats 9
ASCII characters 23
attribute conventions 19
attribute deletion 108
attribute ID 108
attribute inquire 99
attribute length 18, 19, 88, 95, 100, 103, 123
attribute names 18, 95, 96, 97, 99, 100, 103, 108, 110, 112, 116
attribute numbers 108
attribute operations 95
attribute space 95
attribute type 18, 19, 95, 96, 98, 100, 103, 123
attribute values 18, 21, 95, 96, 102, 103
attribute variable ID 100, 102, 108, 109, 112, 113
attributes 18, 21, 30, 32, 33, 95
availability of netCDF software 7

B

backing out of definitions 53
byte CDL type 17
byte constant 124
byte data type 123
byte type 23, 124

C

C, generation of 120
C_format attribute 20
canceling definitions 53
CANDIS 10
CDF Description Language 120
CDL 13, 120
CDL attribute initialization 123
CDL attributes 121, 122
CDL constants 123, 124
CDL data types 122
CDL dimensions 121
CDL example 13, 120
CDL names 14
CDL notation 13, 19
CDL reserved words 23, 122
CDL syntax 120
CDL variable declarations 17
CDL variable initialization 122
CDL variables 121, 122
char CDL type 17
char data type 123
char type 23
character constant 124
character string data 88
character strings 88
character type 124
character-position dimension 88, 89
character-string attributes 88
character-string values 89
closing a netCDF file 32, 47, 51, 53
closing files 30
common netCDF calls 29
computing environments 127
conventional attributes 18, 19
conventions in examples 37
coordinate variables 16, 121
coordinates 16
copying attributes 105
creating a dimension 56
creating a netCDF file 29, 39, 53
creating a variable 66
creating attributes 96

D

data compression 20
data formats 20
data loss 30

NCGOPT 36
NCINQ 32, 33, 49, 50, 61
ncinquire 32, 33, 49, 60
NCLONG 17, 67, 72, 75, 79, 82, 86, 98, 100, 117
NCNOCLOB 39, 40, 46, 48, 57
NCNOWRIT 41, 43, 50, 52, 59, 61, 70, 72
ncopen 31, 32, 34, 41, 43, 49, 51, 53, 58, 60, 63, 69, 71, 74, 78, 81, 85, 88, 92, 97, 99, 102, 105, 108, 110, 112, 116
NCOPN 31, 32, 34, 41, 42, 43, 50, 52, 54, 59, 61, 64, 70, 72, 75, 80, 83, 87, 90, 93, 98, 100, 104, 106, 109, 111, 113, 117
ncopts 35
NCPOPT 36
ncredef 34, 43, 53, 63, 88, 92, 97, 105, 112
NCREDF 34, 43, 53, 54, 64, 90, 93, 98, 106, 113
NCSHORT 17, 67, 72, 75, 79, 82, 86, 98, 100, 117
NCSNC 30, 51, 52
ncsync 30, 51
NCTLEN 116, 117
nctypelen 102, 116
NCUNLIM 57, 67
ncvardef 29, 30, 34, 66, 88
ncvarget 31, 32, 33, 84, 85, 88
ncvarget1 ..... 31, 33, 81
ncvarid 31, 69, 71, 74, 78, 81, 85, 92, 97, 99, 102, 105, 108, 110, 112, 116
ncvarinq 32, 33, 71, 108, 116
ncvarput 29, 30, 34, 77, 78, 88
ncvarput1 ..... 30, 74
ncvarrename 92
NCVDEF 29, 30, 34, 66, 67, 90
NCVERBOS 36
NCVG1C 31, 33, 81
NCVGT 31, 32, 33, 84, 86, 87, 88
NCVGT1 ..... 31, 33, 81, 82, 83
NCVGTC 31, 32, 33, 84, 86, 88, 89
NCVID 31, 69, 70, 72, 75, 80, 83, 87, 93, 98, 100, 104, 106, 109, 111, 113, 117
NCVINQ 32, 33, 71, 72, 109, 117
NCVP1C 74
NCVPT 29, 30, 34, 77, 79, 80, 88
NCVPT1 ..... 30, 74, 75
NCVPTC 29, 30, 34, 77, 88, 89, 90
NCVPTC1 30
NCVREN 92, 93
NCWRITE 41, 54, 64

R

rcode 36

Function and Variable Index

E

error returns 35

M

MAX_NC_NAME 71
MAX_VAR_DIMS 66, 71, 116
MAX_NC_NAME 60, 108
MAXNCNAM 61, 72, 109
MAXVDIMS 67, 72, 117

N

NC_BYTE 17, 66, 71, 96, 116
NC_CHAR 17, 66, 71, 88, 96, 97, 116
NC_CLOBBER 39, 66, 67
NC_DOUBLE 17, 66, 71, 96, 97, 116
NC_FATAL 35
NC_FLOAT 17, 66, 71, 96, 116
NC_GLOBAL 95, 96, 97, 112
NC_LONG 17, 66, 71, 96, 116
NC_NOCLOBBER 39, 47
NC_NOWRITE 43, 49, 58, 69, 71
NC_SHORT 17, 66, 71, 96, 116
nc_type 66, 71, 96, 116
NC_UNLIMITED 13, 66
NC_VERBOSE 35
NC_WRITE 51, 53, 63
NC_BYTE 99
NC_CHAR 99
NC_DOUBLE 99
NC_FLOAT 99
NC_GLOBAL 99, 102, 105, 108
NC_LONG 99
NC_NOCLOBBER 45, 56
NC_NOWRITE 41, 60
NC_SHORT 99
nc_type 99, 102
NC_UNLIMITED 56
NC_WRITE 41, 74
NCABOR 35, 47, 53, 54
ncabort 35, 47, 53
NCACPY 105, 106
NCADEL 112, 113
NCAGT 31, 32, 33, 102, 103, 104
NCAGTC 31, 32, 33, 102, 103, 104
NCAINQ 32, 33, 99, 100, 103, 104
NCANAM 33, 95, 108, 109
NCAPT 29, 30, 34, 96, 97, 98
NCAPTC 29, 30, 34, 96, 97, 98
NCAREN 110, 111
ncattcopy 105
ncattdel 112
ncattget 31, 32, 33, 102
ncattinq 32, 33, 99, 102
ncattname 32, 33, 95, 108
ncattput 29, 30, 34, 96, 97
ncattrename 110
NCBYTE 17, 67, 72, 75, 79, 82, 86, 98, 100, 117
NCCHAR 17, 67, 72, 75, 79, 82, 86, 90, 98, 100, 117
NCCLOB 40
NCCLOS 29, 30, 31, 32, 34, 35, 47, 48, 53
ncclose 29, 30, 31, 32, 34, 35, 47, 53
NCCRE 29, 39, 40, 46, 48, 57, 67
nccreate 29, 39, 45, 47, 56, 66
NCDDEF 29, 30, 34, 54, 56, 57, 67, 90
NCDID 31, 58, 59, 61, 64
ncdimdef 29, 30, 34, 53, 56, 66, 88
ncdimid 31, 58, 60, 63
ncdiminq 32, 33, 60
ncdimrename 63
NCDINQ 32, 33, 60, 61
NCDOUBLE 17, 67, 72, 75, 79, 82, 86, 98, 100, 117
NCDREN 63, 64
ncdump 13, 119, 123, 126
ncendef 29, 30, 34, 35, 45, 47, 51, 63, 88, 92, 97, 105, 112
NCENDF 29, 30, 34, 35, 45, 46, 47, 51, 64, 90, 93, 98, 106, 113
ncerr 35
NCFATAL 36
NCFLOAT 17, 67, 72, 75, 79, 82, 86, 98, 100, 117
ncgen 13, 119, 120, 123, 126
NCGLOBAL 98, 100, 104, 106, 109, 113

```
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
CHARACTER*(*) ATTNAME   ! attribute name
INTEGER ATTLEN          ! number of elements in an attribute vector
INTEGER INCDFID         ! input netCDF ID
INTEGER INVARID         ! input variable ID
INTEGER OUTCDFID        ! output netCDF ID
INTEGER OUTVARID        ! output variable ID
INTEGER ATTNUM          ! attribute number
CHARACTER*(*) NEWNAME   ! new attribute name
INTEGER VARIDS(*)       ! IDs of variables in netCDF link subset
INTEGER DIMINS(*)       ! lower limit of indices in hyperslab of link
INTEGER DIMAXS(*)       ! upper limit of indices in hyperslab of link
INTEGER LNCDFID         ! netCDF ID of link
```

13. 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(CDFID, rcode)
SUBROUTINE NCENDF(CDFID, rcode)
SUBROUTINE NCCLOS(CDFID, rcode)
SUBROUTINE NCINQ(CDFID, ndims,nvars,natts,recdim,rcode)
SUBROUTINE NCSNC(CDFID, rcode)
SUBROUTINE NCABOR(CDFID, rcode)
INTEGER FUNCTION NCDDEF(CDFID,DIMNAME,SIZE, rcode)
INTEGER FUNCTION NCDID(CDFID,DIMNAME, rcode)
SUBROUTINE NCDINQ(CDFID,DIMID, dimname,size,rcode)
SUBROUTINE NCDREN(CDFID,DIMID,DIMNAME, rcode)
INTEGER FUNCTION NCVDEF(CDFID,VARNAME,DATATYPE,NVDIMS,VDIMS, rcode)
INTEGER FUNCTION NCVID(CDFID,VARNAME, rcode)
SUBROUTINE NCVINQ(CDFID,VARID, varname,datatype,nvdims,vdims,nvatts,rcode)
SUBROUTINE NCVPT1(CDFID,VARID,INDICES,VALUE, rcode)
SUBROUTINE NCVP1C(CDFID,VARID,INDICES, CHVAL, rcode)
SUBROUTINE NCVGT1(CDFID,VARID,INDICES, value, rcode)
SUBROUTINE NCVG1C(CDFID,VARID,INDICES, chval, rcode)
SUBROUTINE NCVPT(CDFID,VARID,START,COUNTS,VALUE, rcode)
SUBROUTINE NCVPTC(CDFID,VARID,START,COUNTS,STRING,LENSTR, rcode)
SUBROUTINE NCVGT(CDFID,VARID,START,COUNTS, value,rcode)
SUBROUTINE NCVGTC(CDFID,VARID,START,COUNTS, string,LENSTR,rcode)
SUBROUTINE NCVREN(CDFID,VARID,VARNAME, rcode)
SUBROUTINE NCAPT(CDFID,VARID,ATTNAME,DATATYPE,ATTLEN,VALUE, rcode)
SUBROUTINE NCAPTC(CDFID,VARID,ATTNAME,DATATYPE,LENSTR,STRING, rcode)
SUBROUTINE NCAINQ(CDFID,VARID,ATTNAME, datatype,attlen,rcode)
SUBROUTINE NCAGT(CDFID,VARID,ATTNAME, value,rcode)
SUBROUTINE NCAGTC(CDFID,VARID,ATTNAME, string,LENSTR,rcode)
SUBROUTINE NCACPY(INCDFID,INVARID,ATTNAME,OUTCDFID,OUTVARID, rcode)
SUBROUTINE NCANAM(CDFID,VARID,ATTNUM, attname,rcode)
SUBROUTINE NCAREN(CDFID,VARID,ATTNAME,NEWNAME, rcode)
SUBROUTINE NCADEL(CDFID,VARID,ATTNAME, rcode)
INTEGER FUNCTION NCTLEN(DATATYPE, rcode)
SUBROUTINE NCPOPT(NCOPTS)
SUBROUTINE NCGOPT(ncopts)

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 CDFID           ! netCDF ID, returned by NCCRE or NCOPN
INTEGER NDIMS           ! number of dimensions in netCDF
```

12. Summary of C Interface

```
int nccreate(char* path,int cmode);
int ncopen(char* path,int mode);
int ncredef(int cdfid);
int ncendef(int cdfid);
int ncclose(int cdfid);
int ncinquire(int cdfid,int* ndims,int* nvars,int* natts,int* recdim);
int ncsync(int cdfid);
int ncabort(int cdfid);
int ncdimdef(int cdfid,char* name,int length);
int ncdimid(int cdfid,char* name);
int ncdiminq(int cdfid,int dimid,char* name,int* length);
int ncdimrename(int cdfid,int dimid,char* name);
int ncvardef(int cdfid,char* name,nc_type datatype,int ndims,int dim[]);
int ncvarid(int cdfid,char* name);
int ncvarinq(int cdfid,int varid,char* name,nc_type* datatype,int* ndims,
              int dim[],int* natts);
int ncvarput1(int cdfid,int varid,int coords[],void* value);
int ncvarget1(int cdfid,int varid,int coords[],void* value);
int ncvarput(int cdfid,int varid,int start[],int count[],void* value);
int ncvarget(int cdfid,int varid,int start[],int count[],void* value);
int ncvarrename(int cdfid,int varid,char* name);
int ncattput(int cdfid,int varid,char* name,nc_type datatype,int len,
             void* value);
int ncattinq(int cdfid,int varid,char* name,nc_type* datatype,int* len);
int ncattget(int cdfid,int varid,char* name,void* value);
int ncattcopy(int incdf,int invar,char* name,int outcdf,int outvar);
int ncattname(int cdfid,int varid,int attnum,char* name);
int ncattrename(int cdfid,int varid,char* name,char* newname);
int ncattdel(int cdfid,int varid,char* name);
int nctypelen(nc_type datatype);
```

11. How to Obtain netCDF Software

The current version of netCDF software is available using anonymous FTP.

The netCDF C interface has been compiled and tested successfully on a Sun 3 and SPARCstation under SunOS, a DEC VAX under VMS, a DEC VAX and DECstation 3100 under Ultrix, an Apple Macintosh II under MacOS, a Compaq 80386 under 386-ix, a NeXT under MACH, a Stellar under Stellix , and a Cray XMP under UNICOS. Since there is no standard for FORTRAN to C interfaces, the set of FORTRAN jackets that provide the FORTRAN interface on top of the C library varies for each machine architecture and set of compilers. Consequently, the FORTRAN interface has so far only been tested successfully on a Sun 3 and SPARCstation under SunOS, a DEC VAX under VMS, a DEC VAX under Ultrix, and a DECstation 3100 under Ultrix.

Included in the software distribution are: the C source for the netCDF data access library, C source for the FORTRAN jacket library for UNIX and VMS FORTRAN compilers, source for the netCDF utilities ncdump and ncgen, a directory of test programs to verify the correct implementation of the netCDF library in new environments, and XDR source code for environments that do not yet support XDR. Instructions for porting the netCDF software to a new environment are also included.

For UNIX systems, a compressed tar file can be accessed (in binary mode) from the file `netcdf.tar.Z` in the anonymous FTP directory of `unidata.ucar.edu`. VMS sites can get a backup saveset of the same software from the anonymous FTP directory of `laurel.ucar.edu`. The software distribution includes a PostScript file of the netCDF User's Guide.

A mailing list, `netcdfgroup@unidata.ucar.edu`, is available for discussion of the netCDF interface and announcements about netCDF bugs, fixes, and enhancements. To subscribe, send a request to `netcdfgroup-adm@unidata.ucar.edu`.

10.2 ncdump

`ncdump` is a tool that generates an ASCII representation of a netCDF file, 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.

UNIX syntax for invoking `ncdump`:

`ncdump [ -h] [ -c] [ -n` *name*`] [`*inputfile*`]`

where:

`-h` Produce only the "header" information in the output file; that is, the declarations of dimensions and variables but no data values for the variables.

`-c` Produce the "header" information in the output file and the data values for coordinate variables (variables that are also dimensions).

`-n name` Specify a different name for the CDL description than the default. CDL requires a name for a CDL description that is used by `cdfgen` to generate the file name for an output netCDF file. By default this name is constructed from the last component of the pathname of the input file by stripping off any extension it has (conventionally, `.cdf` is used as an extension for netCDF files). Use this option to specify a different name.

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
```

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. 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
```

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, except that if 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, except that if it begins with `0x`, it is interpreted as a hexadecimal constant. Examples of valid `long` constants include:

```
-2
1234567890L
0123            // octal
0x7ff           // hexadecimal
```

Except for the added data-type `byte` and the lack of the type qualifier `unsigned`, CDL 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. In declarations, type names may be specified in either upper or lower case.

`Byte`s 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 in output FORTRAN code.

`Short`s 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.

`Long`s 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. `int` and `integer` are accepted as synonyms for `long` in CDL declarations.

`Float`s 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. `real` is accepted as a synonym for `float` in CDL declarations.

`Double`s 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.

CDL Notation for Data Constants

This section explains the current CDL notation for netCDF constants. This remains one of the more volatile parts of the netCDF software. Neither CDL nor the utility programs that use it (`ncdump` and `ncgen`) are part of the netCDF procedural interface yet, and so the information presented here is subject to change.

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

as `degrees 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 metadata (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.

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.

CDL Data Types

The CDL data types are:

`char`	Character strings.
`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).

```
netcdf foo {    // example netCDF specification in CDL

dimensions:
lat = 9, 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 above sea level";
  z:valid range = 0., 5000.;
  p:missing value = -9999.;
  rh:missing value = -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

10.1 ncgen

`ncgen` is a tool that generates a netCDF file or the C or FORTRAN programs required to create the netCDF file. The input to `ncgen` is a description of a netCDF file in a tiny language known as CDL (netCDF description language). 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 [ -n] [ -o` *outputfile*`] [ -c] [ -f] [`*inputfile*`]`

where:

`-n` Create a 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 `.cdf` 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 `-n` option. (This option is necessary because netCDF files are random access files created with `fseek()` calls, and hence cannot be written to the 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.

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`, `missing_value`), and some data.

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 the higher-level netCDF operations and generic applications for processing netCDF files. Currently there are only a few such higher-level operations available. Below we describe `ncgen` and `ncdump`, two tools for converting between binary netCDF files and an ASCII representation of netCDF files.

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.cdf`':

```
      INCLUDE 'netcdf.inc'
        ...
      INTEGER  CDFID              ! 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
        ...
      CDFID = NCOPN ('foo.cdf', NCNOWRIT, RCODE)
        ...
      RHID = NCVID (CDFID, 'rh', RCODE)
* get type of "rh"
      CALL NCVINQ (CDFID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT,
     +             RCODE)
      RHBYTS = NCTLEN (RHTYPE)
```

9.1 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`. Possible causes of errors include

- 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.cdf`':

```
#include "netcdf.h"
   ...
int  cdfid;       /* 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" */
   ...
cdfid = ncopen("foo.cdf", NC_NOWRITE);
   ...
rh_id = ncvarid (cdfid, "rh");
/* get type.  we don't need name, since we already know it */
ncvarinq (cdfid, rh_id, (char *) 0, &rh_type, &rh_ndims, rh_dims,
          &rh_natts);
rhbytes = nctypelen (rh_type);
```

9. Miscellaneous netCDF Operations

Other miscellaneous operations supported by this interface include

- Get number of bytes for a given data type.

NCADEL: FORTRAN Interface

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

CDFID	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	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.cdf':

```
      INCLUDE 'netcdf.inc'
        ...
      INTEGER  CDFID     ! netCDF ID
      INTEGER  RHID      ! variable ID
        ...
      CDFID = NCOPN ('foo.cdf', NCWRITE, RCODE)
        ...
      RHID = NCVID (CDFID, 'rh', RCODE)
        ...
* delete attribute
      CALL NCREDF (CDFID)  ! enter define mode
      CALL NCADEL (CDFID, RHID, 'Units', RCODE)
      CALL NCENDF (CDFID)  ! leave define mode
```

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 cdfid, int varid, char* name);
```

`cdfid`	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`	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.cdf`':

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

NCAREN: FORTRAN Interface

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

CDFID	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	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.cdf':

```
      INCLUDE "netcdf.inc"
         ...
      INTEGER  CDFID     ! netCDF ID
      INTEGER  RHID      ! variable ID
         ...
      CDFID = NCOPN ("foo.cdf", NCNOWRIT, RCODE)
         ...
      RHID = NCVID (CDFID, "rh", RCODE)
         ...
* rename attribute
      CALL NCAREN (CDFID, RHID, "units", "Units", 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 cdfid, int varid, char* name, char* newname);
```

`cdfid`	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`	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.cdf`':

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

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

NCANAM: FORTRAN Interface

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

CDFID	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.cdf':

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

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 as an attribute of that variable. 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 cdfid, int varid, int attnum, char* name);
```

`cdfid` 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 so 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.cdf`':

```
      INCLUDE 'netcdf.inc'
        ...
      INTEGER  CDFID1, CDFID2     ! netCDF IDs
      INTEGER  RHID, AVRHID       ! variable IDs
        ...
      CDFID1 = NCOPN ('foo.cdf', NCNOWRIT, RCODE)
      CDFID2 = NCOPN ('bar.cdf', NCWRITE, RCODE)
        ...
      RHID = NCVID (CDFID1, 'rh', RCODE)
      AVRHID = NCVID (CDFID2, 'avgrh', RCODE)
        ...
      CALL NCREDF (CDFID2)  ! enter define mode
* copy variable attribute from "rh" to "avgrh"
      CALL NCACPY (CDFID1, RHID, 'units', CDFID2, AVRHID, RCODE)
      CALL NCACPY (CDFID, NCGLOBAL, 'title', TITLE)
        ...
      CALL NCENDF (CDFID2)  ! leave define mode
```

```
#include "netcdf.h"
   ...
int  cdfid1, cdfid2;       /* netCDF IDs */
int  rh_id, avgrh_id;      /* variable IDs */
   ...
cdfid1 = ncopen("foo.cdf", NC_NOWRITE);
cdfid2 = ncopen("bar.cdf", NC_WRITE);
   ...
rh_id = ncvarid (cdfid1, "rh");
avgrh_id = ncvarid (cdfid2, "avgrh");
   ...
ncredef(cdfid2);           /* enter define mode */
/* copy variable attribute from "rh" to "avgrh" */
ncattcopy(cdfid1, rh_id, "units", cdfid2, avgrh_id);
   ...
ncendef(cdfid2);           /* 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.cdf`' to the variable `avgrh` in another existing netCDF file named '`bar.cdf`', assuming that the variable `avgrh` already exists, but does not yet have a `units` attribute:

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, 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.cdf`' to the variable `avgrh` in another existing netCDF file named '`bar.cdf`', assuming that the variable `avgrh` already exists, but does not yet have a `units` attribute:

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 actual string length of the STRING parameter in the caller.

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.cdf'. 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  CDFID, 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
        ...
      CDFID = NCOPN ('foo.cdf', NCWRITE, RCODE)
        ...
      RHID = NCVID (CDFID, 'rh', RCODE) ! get ID
        ...
* find out attribute lengths, to make sure we have enough space
      CALL NCAINQ (CDFID, RHID, 'valid_range', VRTYPE, VRLEN,
     +             RCODE)
      CALL NCAINQ (CDFID, 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 (CDFID, RHID, 'valid_range', VRVAL, RCODE)
      ENDIF
      IF (TLEN > MTLEN) THEN
          WRITE (*,*) 'title attribute too big!'
          CALL EXIT
      ELSE
          CALL NCAGTC (CDFID, NCGLOBAL, 'title', TITLE, MTLEN, RCODE)
      ENDIF
```

```
#include "netcdf.h"
   ...
int  cdfid;                /* 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 */

   ...
cdfid = ncopen("foo.cdf", NC_NOWRITE);
   ...
rh_id = ncvarid (cdfid, "rh");
   ...
/* find out how much space is needed for attribute values */
ncattinq (cdfid, rh_id, "valid_range", &vr_type, &vr_len);
ncattinq (cdfid, 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(cdfid, rh_id, "valid_range", (void *)vr_val);
ncattget(cdfid, NC_GLOBAL, "title", (void *)title);
   ...
```

NCAGT, NCAGTC: FORTRAN Interface

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

      SUBROUTINE NCAGTC (INTEGER CDFID, 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.

`CDFID` 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 the user must provide enough space to hold them. If you don't know how much

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 cdfid, int varid, char* name, void* value);
```

`cdfid` 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.cdf`'. In this example, it is assumed that we don't know how many values will be returned, but that we do know the types of the attributes. Hence, to allocate enough space to store them, we must first inquire about the length of the attributes.

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

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

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

NCAINQ: FORTRAN Interface

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

`CDFID` 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.cdf`':

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 cdfid, int varid, char* name,
             nc_type* datatype, int* len);
```

`cdfid` 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`.

`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).

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.cdf`':

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 actual string length of the `STRING` parameter.

`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.cdf`':

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

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.**

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.cdf`':

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

NCAPT, NCAPTC: FORTRAN Interface

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

      SUBROUTINE NCAPTC (INTEGER CDFID, 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.

`CDFID` 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

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 cdfid, int varid, char* name, nc_type datatype,
             int len, void* values);
```

`cdfid` 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 section 2.3.1 [Attribute Conventions], page 19, 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

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 it 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.

NCVREN: FORTRAN Interface

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

`CDFID` 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.cdf`':

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

7.10 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 cdfid, int varid, char* name);
```

`cdfid`	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.cdf`':

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

7.9 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* that is a particular value at one end of the range of each netCDF type used to fill in any missing values when a netCDF variable is first written. It is also possible to use your own fill value instead, if the default fill value is a useful value for the variable, or to ignore fill values and use the entire range of each netCDF type.

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 regular 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 constant fill values for each type are defined in the include file '`netcdf.h`' (or '`netcdf.inc`' for FORTRAN). Fill values are used for filling in missing data whenever a value is put beyond the end of data that has already been written. The fill values have no other special meaning, so they can be used for valid values if there is no need for fill values, or if alternate fill values are used. To use an alternate fill value for a variable, just initialize the variable with the fill value before you begin writing data to it. If you do this, you should also define an associated variable attribute `missing_value` with the appropriate type and value to capture your intent.

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.

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.cdf' 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  CDFID, 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
        ...
      CDFID = NCOPN('foo.cdf', NCWRITE, RCODE)
      CALL NCREDF(CDFID) ! enter define mode
        ...
* define character-position dimension for strings of max length 40
      CHID = NCDDEF(CDFID, "chid", 40, RCODE)
        ...
* define a character-string variable
      TXDIMS[1] = CHID   ! character-position dimension first
      TXDIMS[2] = TIMEID
      TXID = NCVDEF(CDFID, "tx", NCCHAR, TDIMS, TXDIMS, RCODE)
        ...
      CALL NCENDF(CDFID) ! 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 (CDFID, TXID, TSTART, TCOUNT, TXVAL, 40, RCODE)
```

and data are to be added to an existing netCDF file named '`foo.cdf`' that already has an unlimited record dimension `time`.

```
#include "netcdf.h"
   ...
int  cdfid;           /* 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 */
int tx_start[TDIMS];
int tx_count[TDIMS];
static char tx_val[] =
        "example string"; /* string to be put */
   ...
cdfid = ncopen("foo.cdf", NC_WRITE);
ncredef(cdfid);       /* enter define mode */
   ...
/* define character-position dimension for strings of max length 40 */
chid = ncdimdef(cdfid, "chid", 40);
   ...
/* define a character-string variable */
tx_dims[0] = timeid;
tx_dims[1] = chid;    /* character-position dimension last */
tx_id = ncvardef (cdfid, "tx", NC_CHAR, TDIMS, tx_dims);
   ...
ncendef(cdfid);       /* 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(cdfid, 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

7.8 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. 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

`LENSTR` For `NCVGTC`, the declared length of the `STRING` argument. This should be at least as large as the product of the elements of the `COUNT` vector.

`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.cdf`'. 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  CDFID, 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 /TIMES, LATS, LONS/
   ...
CDFID = NCOPN ('foo.cdf', NCNOWRIT, RCODE)
   ...
RHID = NCVID (CDFID, 'rh', RCODE)! get ID
CALL NCVGT (CDFID, RHID, START, COUNT, RHVALS, RCODE)
```

NCVGT, NCVGTC: FORTRAN Interface

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

 SUBROUTINE NCVGTC(INTEGER CDFID, 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 is for reading numeric values from a variable of numeric type, and the second is for reading character values from a variable of character type.

`CDFID` 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 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 if 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 into the specified hyperslab is with the first dimension varying fastest (like the FORTRAN convention). The data may be of a type corresponding to the netCDF types `NCCHAR` or `NCBYTE`.

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 specified hyperslab is with the last dimension 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 if 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.cdf`'. 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  cdfid;                         /* netCDF ID */
int  rh_id;                         /* variable ID */
static int start[] = {0, 0, 0};     /* start at first value */
static int count[] = {TIMES, LATS, LONS};
double rh_vals[TIMES*LATS*LONS];    /* array to hold values */
   ...
cdfid = ncopen("foo.cdf", NC_NOWRITE);
   ...
rh_id = ncvarid (cdfid, "rh");
   ...
/* read hyperslab of values from netCDF variable */
ncvarget(cdfid, rh_id, start, count, (void *) rh_vals);
```

7.7 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 cdfid, int varid, int start[], int count[],
             void *values);
```

`cdfid` 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 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 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

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.cdf`'. 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  CDFID, RCODE
INTEGER  RHID               ! variable ID
INTEGER  RHINDX(3)          ! where to get value
DOUBLE PRECISION RHVAL      ! put it here
DATA RHINDX /4, 3, 2/
   ...
CDFID = NCOPN ('foo.cdf', NCNOWRIT, RCODE)
   ...
RHID = NCVID (CDFID, 'rh', RCODE)! get ID
CALL NCVGT1 (CDFID, RHID, RHINDX, RHVAL, RCODE)
```

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

NCVGT1: FORTRAN Interface

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

 SUBROUTINE NCVG1C (INTEGER CDFID, 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 is for reading a numeric value in a variable of numeric type, and the second is for reading a character value in a variable of character type.

`CDFID` 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 if 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.

7.6 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 cdfid, int varid, int mindex[], void *value);
```

`cdfid` 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 if 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.cdf`'. 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:

`LENSTR` For `NCVPTC`, the declared length of the `STRING` argument. This should be at least as large as the product of the elements of the `COUNT` vector.

`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.cdf`'. 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  CDFID, 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/
         ...
      CDFID = NCOPN ('foo.cdf', NCWRITE, RCODE)
         ...
      RHID = NCVID (CDFID, '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 (CDFID, RHID, START, COUNT, RHVALS, RCODE)
```

NCVPT: FORTRAN Interface

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

 SUBROUTINE NCVPTC(INTEGER CDFID, 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 is for writing numeric values into a variable of numeric type, and the second is for writing character values into a variable of character type.

`CDFID` 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 if the wrong type of data is used.**

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

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 into the specified hyperslab is with the last dimension 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 if 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.cdf`'. 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  cdfid;                          /* netCDF ID */
int  rh_id;                          /* variable ID */
static int start[] = {0, 0, 0};      /* start at first value */
static int count[] = {TIMES, LATS, LONS};
double rh_vals[TIMES*LATS*LONS];     /* array to hold values */
int i;
   ...
cdfid = ncopen("foo.cdf", NC_WRITE);
   ...
rh_id = ncvarid (cdfid, "rh");
   ...
for (i = 0; i < TIMES*LATS*LONS; i++)
    rh_vals[i] = 0.5;
/* write hyperslab of values into netCDF variable */
ncvarput(cdfid, rh_id, start, count, (void *) rh_vals);
```

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 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 cdfid, int varid, int start[], int count[],
             void *values);
```

`cdfid` 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 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 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

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  CDFID, RCODE
INTEGER  RHID               ! variable ID
INTEGER  RHINDX(3)          ! where to put value
DATA RHINDX /4, 3, 2/
   ...
CDFID = NCOPN ('foo.cdf', NCWRITE, RCODE)
   ...
RHID = NCVID (CDFID, 'rh', RCODE)  ! get ID
CALL NCVPT1 (CDFID, RHID, RHINDX, 0.5, RCODE)
```

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

NCVPT1: FORTRAN Interface

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

 SUBROUTINE NCVP1C (INTEGER CDFID, 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 is for putting a numeric value in a variable of numeric type, and the second is for putting a character value in a variable of character type.

`CDFID` 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 if 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.cdf`'. For simplicity in this example, we assume that we know

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 cdfid, int varid, int mindex[], void *value);
```

`cdfid` 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 if 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.cdf`'. 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:

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

```
      INCLUDE 'netcdf.inc'
         ...
      INTEGER  CDFID, 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
         ...
      CDFID = NCOPN ('foo.cdf', NCNOWRIT, RCODE)
         ...
      RHID = NCVID (CDFID, 'rh', RCODE)! get ID
      CALL NCVINQ (CDFID, RHID, RHNAME, RHTYPE, RHN, RHDIMS, RHNATT,
     +             RCODE)
```

file named 'foo.cdf':

```
#include "netcdf.h"
   ...
int  cdfid;                        /* 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 */
   ...
cdfid = ncopen("foo.cdf", NC_NOWRITE);
   ...
rh_id = ncvarid (cdfid, "rh");
/* we don't need name, since we already know it */
ncvarinq (cdfid, rh_id, (char *) 0, &rh_type, &rh_ndims, rh_dims,
          &rh_natts);
```

NCVINQ: FORTRAN Interface

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

`CDFID` 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.

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 cdfid, int varid, char* name, nc_type* datatype,
             int* ndims, int dim[], int* natts);
```

`cdfid` 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`.

`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.

`dim` 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`.

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

Here is an example using `ncvarinq` to find out about a variable named `rh` in an existing netCDF

`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.cdf`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER  CDFID, RCODE
INTEGER  RHID                    ! variable ID
   ...
CDFID = NCOPN ('foo.cdf', NCNOWRIT, RCODE)
   ...
RHID = NCVID (CDFID, 'rh', 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 cdfid, char* name);
```

`cdfid` 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.cdf`':

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

NCVID: FORTRAN Interface

```
      INTEGER FUNCTION NCVID(INTEGER CDFID,
     +                       CHARACTER*(*) VARNAM,
     +                       INTEGER RCODE)
```

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

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

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

NCVDEF: FORTRAN Interface

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

`CDFID` 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.cdf`':

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 cdfid, char* name, nc_type datatype,
             int ndims, int dim[]);
```

`cdfid` 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`.

`dim` 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.cdf`':

7. Variables

Variables for a netCDF file are defined when it is created, while the netCDF file is in define mode. Additional 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, special 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.
- 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.
- Rename a variable.

NCDREN: FORTRAN Interface

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

CDFID	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.cdf':

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

6.4 Rename a Dimension

The function `ncdimrename` (or `NCDREN` for FORTRAN) renames an existing dimension in a netCDF 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 cdfid, int dimid, char* name);
```

`cdfid`	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.cdf`':

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

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

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

NCDINQ: FORTRAN Interface

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

`CDFID` 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.cdf`':

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 cdfid, int dimid, char* name, int* size);
```

`cdfid` 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.

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.cdf`':

NCDID: FORTRAN Interface

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

`CDFID` 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.cdf`':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER CDFID, RCODE, LATID
   ...
CDFID = NCOPN('foo.cdf', NCNOWRIT, RCODE)
   ...
LATID = NCDID(CDFID, 'lat', 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 cdfid, char* name);
```

`cdfid` 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.cdf`':

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

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

NCDDEF: FORTRAN Interface

```
      SUBROUTINE NCDDEF (INTEGER CDFID,
     +                   CHARACTER*(*) DIMNAM,
     +                   INTEGER DIMSIZ,
     +                   INTEGER RCODE)
```

`CDFID` 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.cdf`':

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

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 cdfid, char* name, int size);
```

`cdfid` 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 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.cdf`':

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 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, so that generic applications 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.

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

NCABOR: FORTRAN Interface

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

`CDFID` 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.cdf`':

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

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 cdfid);
```

`cdfid` 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.cdf`':

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

NCSNC: FORTRAN Interface

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

`CDFID` 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.cdf`':

```
      INCLUDE 'netcdf.inc'
        ...
      INTEGER CDFID, RCODE
        ...
      CDFID = NCOPN('foo.cdf', NCNOWRIT, RCODE)
        ...
* write data or change attributes
        ...
      CALL NCSNC(CDFID, 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. 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.

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 cdfid);
```

`cdfid` 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.cdf`':

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

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

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

NCINQ: FORTRAN Interface

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

CDFID	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.cdf':

```
INCLUDE 'netcdf.inc'
   ...
INTEGER CDFID, NDIMS, NVARS, NATTS, RECDIM, RCODE
   ...
CDFID = NCOPN('foo.cdf', NCNOWRIT, RCODE)
   ...
CALL NCINQ(CDFID, NDIMS, NVARS, NATTS, RECDIM, 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 variable ID of the dimension defined with unlimited size, if any.

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

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

ncinquire: C Interface

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

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

`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`.

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

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

NCCLOS: FORTRAN Interface

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

`CDFID` 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.cdf`' and release its netCDF ID:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER CDFID, RCODE
   ...
CDFID = NCCRE('foo.cdf', NCNOCLOB, RCODE)

   ...  ! create dimensions, variables, attributes

CALL NCCLOS(CDFID, 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 cdfid);
```

`cdfid` 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.cdf`' and release its netCDF ID:

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

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

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

`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.cdf`' and put it into data mode:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER CDFID
   ...
CDFID = NCCRE('foo.cdf', NCNOCLOB, RCODE)

   ...  ! create dimensions, variables, attributes

CALL NCENDF(CDFID, 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.

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 cdfid);
```

`cdfid` 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.cdf`' and put it into data mode:

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

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

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

NCENDF: FORTRAN Interface

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

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

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

```
INCLUDE 'netcdf.inc'
   ...
INTEGER CDFID
   ...
CDFID = NCOPN('foo.cdf', NCNOWRIT, RCODE)
   ...
CALL NCREDF(CDFID, 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 cdfid);
```

`cdfid` 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.cdf`' and put it into define mode:

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

NCREDF: FORTRAN Interface

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

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

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

`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.cdf`' for reading:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER CDFID
   ...
CDFID = NCOPN('foo.cdf', NCNOWRIT, 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(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.cdf`' for reading:

```
#include "netcdf.h"
   ...
int cdfid;
   ...
cdfid = ncopen("foo.cdf", 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.

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.

Here is an example of the creation of a netCDF file named '`foo.cdf`', assuming we want the file to be created in the current directory only if it does not already exist:

```
INCLUDE 'netcdf.inc'
   ...
INTEGER CDFID
   ...
CDFID = NCCRE('foo.cdf', NCNOCLOB, RCODE)
```

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 (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 '`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.

Here is an example of the creation of a netCDF file named '`foo.cdf`'; we want the file to be created in the current directory only if it does not already exist:

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

- 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 CDFID       ! 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. 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 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.

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,

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`'.

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. All the examples in this document assume this default error-handling behavior, so the examples include no checking of error returns.

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;
```

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 */
  ...
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:

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

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

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 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, or `ncvarget` (or `NCVGT` or `NCVGTC`) for aggregates of values using hyperslab access.

Next, a call to `ncopen` (or `NCOPN`) 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, or `ncvarget` (or `NCVGT` or `NCVGTC`) for hyperslabs 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:

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 what the dimensions, variables, and attributes are. 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:

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

```
      ...
   CALL NCAPT or NCAPTC ! attribute put: assign attribute values
      ...
CALL NCENDF             ! end definitions: leave define mode
      ...
CALL NCVPT or NCVPTC    ! variable put: provide values for variables
  ...
CALL NCAPT or NCAPTC    ! attribute put: change attribute values
  ...
CALL 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 `NCVPTC1`); while arbitrary hyperslabs of data are written using `ncvarput` (or `NCVPT` or `NCVPTC`) instead.

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. Use of the netCDF Library

It is not necessary to know about all the netCDF modules 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 the typical 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 is as follows, where 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:

```
CALL NCCRE              ! create netCDF file: enter define mode
    ...
   CALL NCDDEF          ! define dimensions: from name and size
    ...
   CALL NCVDEF          ! define variables: from name, type, dimensions
```

netCDF library won't provide much help or hindrance with constructing such data structures, but it is possible to use attributes to name associated index variables. For example, a variable attribute such as '`array_index_var = "v_index"`' attached to one variable may provide the name of another associated variable to be used as an index for fast retrieval by value.

```
        C                FORTRAN

z[0][1][0][0]       z(1, 1, 2, 1)
z[0][1][0][1]       z(2, 1, 2, 1)
z[0][1][0][2]       z(3, 1, 2, 1)
z[0][1][0][3]       z(4, 1, 2, 1)

   ...                  ...

z[2][1][4][7]       z( 8, 5, 2, 3)
z[2][1][4][8]       z( 9, 5, 2, 3)
z[2][1][4][9]       z(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.

To perform conventional record-oriented access, you specify a netCDF file, a record variable (one defined with an unlimited dimension), and use the record number as the value of the first dimension (last dimension in FORTRAN), using hyperslab access to get the record of values. When efficiency is a concern, you should keep in mind the order in which netCDF data is written on the disk, since the best I/O performance is achieved by reading or writing contiguous data. All variable data is 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 multidimensional variables with attached vector attributes. The 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

In C, this corresponds to

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

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

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

using a multidimensional array declaration.

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, as in

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

Then 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:

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 direct (random) access to single data values, direct access to an arbitrary hyperslab of data for a single variable, and record-oriented access to data for a single variable (as a special case of hyperslab access) in an open netCDF file.

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 was 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 kind of 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 first example netCDF you wish to read all the data for the `z` variable at the second (500-mb) 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 `z` is declared as

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

in the CDL notation.

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` used for 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 data-type `byte` and the lack of `unsigned`, 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.

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

Note that more 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.

2.3.2 Differences between Attributes and Variables

In contrast to variables, which are intended for data, attributes are intended for metadata. 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 metadata 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 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.

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 is 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 for low-resolution data to be stored as small integers in a netCDF file. When scaled data is written, the application should first subtract the offset and then divide by the scale factor.

`missing_value`
: If present for a variable, this value is considered to be a special value that indicates missing data. Hence an application that is displaying the data should ignore all data points with this value. The missing value should be outside the range specified by `valid_range` for a variable. It is not necessary to define your own `missing_value` attribute for a variable if the default *fill value* for the type of the variable is adequate. See section 7.9 [Missing Values], page 91. If you define your own `missing_value` attribute for a floating point type, use an integer less than 1000000 in absolute value to avoid machine precision problems.

`C_format` A character array for the format that should be used to print values for this variable by C applications.

`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.

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.

In the example netCDF, `units` is an attribute for the variable `lat` that has a length 13-character array value '`degrees north`'. `valid_range` is an attribute for the variable `t` that has length 2 and values '`-100.0`' and '`100.0`'.

Two global attributes—`source` and `base_time`—are defined for the example netCDF. Both are character arrays 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 will, by convention, expect certain variable and global attributes. If you want to be able to use these generic applications with your files, you should use the following conventional names for these commonly used attributes:

`units` A character array giving 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 is compiling a suggested standard for data in the atmospheric sciences.

`long_name`
A long descriptive name 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

2.3 Attributes

A netCDF *attribute* is meant to contain information about a netCDF variable or about an entire netCDF file. This information is *metadata*, 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 a generic units-converter program, the values of the `units` attributes should be expressed in a conventional form as a character string that can be interpreted by that program.

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

Like a dimension name, a variable name is an arbitrary sequence of alphanumeric characters (including _) 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 eight variables. As discussed above, four of these are coordinate variables for dimensions. The remaining variables, `z`, `t`, `p`, and `rh` are meant to contain the "real" data in this netCDF object. 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.

points. In the example netCDF file, variables `z`, `t`, and `p` have exactly the same dimensions, so they refer to different variables defined at the same points on a four-dimensional space-time grid. 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 if it is intended 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 (100, 500, 750, 1000) 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.

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.

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, and
- 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 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

The CDL notation for a netCDF file can be generated automatically by using `ncdump`, a utility program described later. 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. 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 all 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 multidimensional 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 (including the underscore 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.

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:           // all the dimensions are declared first
        lat = 5, lon = 10, level = 4, time = unlimited;

variables:            // variable types, names, and shapes
        float   z(time,level,lat,lon);
        double  p(time,level,lat,lon), t(time,level,lat,lon);
        double  rh(time,lat,lon);
        int     lat(lat), lon(lon), level(level);
        short   time(time);
                      // variable attributes
        lat:units       = "degrees north";
        lon:units       = "degrees east";
        t:long_name     = "temperature";
        t:units         = "degrees Celsius";
        t:valid_range   = -100.0, 100.0;       // min and max
        rh:long_name    = "relative humidity";
        time:units      = "hours from base_time";
                      // global attributes
        :source = "NWS";
        :base_time = "88/10/25 12:00:00";

data:                 // optional data assignments
        level   = 100, 500, 750, 1000;
        lat     = 20, 30, 40, 50, 60;
        lon     = -160,-140,-118,-96,-84,-52,-45,-35,-25,-15;
        time    = 0, 12;
        rh      = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                  .1,.1,.1,.1,.1,.1,.1,.1,.1,.1,
                  .2,.2,.2,.2,.2,.2,.2,.2,.2,.2,
                  .3,.3,.3,.3,.3,.3,.3,.3,.3,.3,
                  .4,.4,.4,.4,.4,.4,.4,.4,.4,.4;
}
```

5. 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.
6. Rew, R. K. and G. P. Davis, "NetCDF: An Interface for Scientific Data Access," *Computer Graphics and Applications*, IEEE, July 1990.
7. 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.
8. Sun Microsystems, "External Data Representation Standard: Protocol Specification," RFC 1014; Information Sciences Institute, May 1988.

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 implementing the software it described. Comparison with alternatives and recent experience in using netCDF are discussed in a conference preprint volume (Rew and Davis, 1990).

1.6 Future Plans for netCDF

We will continue building on the existing netCDF access library for Unidata system software and applications. We intend 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.

A collection of netCDF operators will be made available that provide an algebra of useful operations on generic scientific data stored in netCDF files. These include selectors that will extract subsets of variables or reduce the dimensionality of a netCDF file; constructors that will merge netCDF files, combine variables, or increase dimensionality; graphics generators that 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 will have a wide variety of capabilities.

We plan to extend the netCDF library in an upward-compatible way to support netCDF servers on a network. Applications programs, as clients of the netCDF servers, will be able to access cross-sections of data efficiently, as if it were stored in a local file. A netCDF server will have the ability to support virtual netCDF objects that provide different views of large or remote datasets. Other servers may also be capable of transparently providing a netCDF interface to non-netCDF archives.

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. 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.
4. 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.

The NASA CDF package has been used for many different kinds of data in an extensive collection of applications. It has 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. Its purposes were to explore the possibility of collaborating with NASA to extend the CDF to work with FORTRAN compilers not employing VMS extensions; 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. Coincidentally, this package also used the common data format name, which was later changed to C-Based Analysis and Display System (CANDIS). Unidata learned of Raymond's work, described in (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.

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.

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 in several important ways (Fahle, 1989). Like Raymond's package, the SeaSpace CDF 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), Angel Li of the University of Miami (who had implemented our prototype netCDF on VMS and was a potential user), and Unidata systems development staff. Consensus was reached at the

netCDF interface to access data in inefficient ways: for example, if the user makes a request for a slice of variable data that requires a single value out of each record. This manual discusses characteristics of the netCDF implementation at those points where it is important to know something about how the underlying software works to use the interface effectively.

1.4 Is netCDF a Good Archive Format?

The netCDF can be used as an archive format for storing data, but it will generally take more space than a special-purpose archive format that exploits knowledge of particular characteristics of a set of data. While compression is possible for low-resolution data by using, for example, eight-bit bytes instead of 32-bit floating-point numbers, the 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 metadata (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 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 both Suns and MicroVAXs, with access from both C and FORTRAN software, 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 originated in a paper (Treinish and Gough, 1987) that described 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 developed as a FORTRAN library for VAX/VMS systems to support an abstraction for storing multidimensional scientific data.

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.

Another problem is that DBMSs provide many facilities that are not needed in the analysis, management, and display of scientific data. Elaborate update facilities, concurrency control, audit trails, report writers, and other mechanisms designed for transaction-processing are unnecessary for the applications served by the netCDF interface. The resources and expense required to support these unnecessary facilities cannot be justified for scientific applications.

Other requirements that are difficult to satisfy with existing database systems include support for both C and FORTRAN procedural interfaces, ability to run on both UNIX and VMS systems, and support for an architecture-independent format for representing scientific data.

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-PCs, IBM mainframes, and CRAYs. It assumes only that 8-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. We are currently considering whether a native mode that does not use XDR would be a useful addition to the netCDF implementation. 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 many applications, we consider the overhead of the XDR layer to be 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 from applications, it becomes possible to express simply computations that may require large amounts of computing resources. Furthermore, it may not be obvious which of several ways of expressing a computation through the new interface will make efficient use of computing resources, without understanding something about the implementation. It is certainly possible to use the

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. The 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 that implements this interface is being made freely available to encourage its wide use. The netCDF interface is supported for both C and FORTRAN, and can be used with UNIX, VMS, MSDOS, and MacOS operating systems. Porting the software to other operating systems should not be difficult.

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.

1.2 The netCDF is Not a Database Management System

Why not use an existing database management system (DBMS) for storing scientific data? We have looked at available database packages, both commercial and research-oriented, and have concluded that they are currently inadequate for achieving the goals of the netCDF.

First, most existing DBMSs have poor support for multidimensional objects as the basic unit of data access. 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,

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 UNIX, VMS, MSDOS, and MacOS environments. The FORTRAN interface is currently available for a smaller set of environments (due to the lack of a standard for calling C from FORTRAN).

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

The Network Common Data Form (netCDF) software package described in this manual was developed initially as part of the SDM system and is distributed with it. In essence, the netCDF is the "standard" data-access interface employed by the various SDM software components. The netCDF software, however, may be useful in a wider context. In recognition of this, Unidata provides the software and its documentation separately.

David Fulker, Director
Unidata Program Center

Foreword

Unidata is a national effort, sponsored by the Division of Atmospheric Sciences of the National Science Foundation (NSF), to help universities use computing and communication technologies to access, analyze, and display atmospheric and related data. The program, which is managed by the University Corporation for Atmospheric Research, was begun in 1982 in response to (1) university interests in wider application of interactive processing and display methods to atmospheric science research and education and (2) university concerns about the future availability of and costs for National Weather Service (NWS) data (the distribution modes were being altered by the Automation of Field Operations and Services [AFOS] program). Unidata now provides universities with inexpensive access to near-real-time weather data accompanied with capabilities to select, organize, analyze, and display those data interactively.

The Unidata system represents the results of a partnership between the universities and the Unidata Program Center in Boulder, Colorado. The goal of the program office is to provide a system to the universities, with the capabilities described above, based upon modern communications technology and personal workstations. Universities participate in establishing Unidata policies, specifying broadcast services, developing software, and they provide technical assistance to one another. The program center's role is to coordinate community efforts nationwide, to provide software, and to help universities select and use networked computing systems.

Most Unidata sites are connected to NSFnet, NSF's scientific research network. Such connections provide convenient access to the Program Center for obtaining current copies of software, documentation, announcements, and software updates as well as for reporting problems and requesting assistance.

Unidata distributes a number of software packages. One is an image display and data-handling system developed by the Space Science and Engineering Center (SSEC) of the University of Wisconsin–Madison. This software, called PC-McIDAS (for Man–Computer Interactive Data Access System) runs on IBM PS/2 computers using the OS/2 operating system.

In addition, Unidata is distributing a composite package of data handling, analysis and display software for VMS- and UNIX-based systems, which includes community contributions from several sources. Unidata's goal is to standardize the user interface, the data-access interface, and the display interface, and to adapt existing applications software to match these three. Beyond the suite of capabilities thus provided, the approach maximizes the programmability of the system: users may add new applications at will so long as the interface standards are observed. This composite package is known as Unidata's Scientific Data Management (SDM) system.

Acknowledgments

For helping to make this first release of the Unidata netCDF software possible, we would like to acknowledge the efforts of

- Joe Fahle, SeaSpace, Inc.
- Michael Gough, Apple Computer
- Angel Li, University of Miami
- Dave Raymond, New Mexico Institute of Mining and Technology
- Lloyd Treinish, NASA/NSSDC

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.

netCDF User's Guide

An Interface for Data Access

Version 1.06, June 1990

by Russell K. Rew, Unidata Program Center