Package: ncdf4 (via r-universe)

October 2, 2024

Version 1.23

Date 2024-08-05

Title Interface to Unidata netCDF (Version 4 or Earlier) Format Data Files

Description Provides a high-level R interface to data files written using Unidata's netCDF library (version 4 or earlier), which are binary data files that are portable across platforms and include metadata information in addition to the data sets. Using this package, netCDF files (either version 4 or ``classic" version 3) can be opened and data sets read in easily. It is also easy to create new netCDF dimensions, variables, and files, in either version 3 or 4 format, and manipulate existing netCDF files. This package replaces the former ncdf package, which only worked with netcdf version 3 files. For various reasons the names of the functions have had to be changed from the names in the ncdf package. The old ncdf package is still available at the URL given below, if you need to have backward compatibility. It should be possible to have both the ncdf and ncdf4 packages installed simultaneously without a problem. However, the ncdf package does not provide an interface for netcdf version 4 files.

SystemRequirements netcdf library version 4.1 or later

License GPL (>= 3)

URL https://cirrus.ucsd.edu/~pierce/ncdf/

NeedsCompilation yes

Author David Pierce [aut, cre]
(<https://orcid.org/0000-0002-2453-9030>)

Maintainer David Pierce <dpierce@ucsd.edu>

Repository CRAN

Date/Publication 2024-08-17 21:30:02 UTC

Contents

`ncdf4-package` *Read, write, and create netCDF files (including version 4 format)*

Description

Read from or write to existing netCDF format files, or create new ones.

Details

More information on this package, including detailed installation instructions, can be found at http://dwpierce.com/software.

The netCDF data file format from Unidata is a platform-independent, binary file that also contains metadata describing the contents and format of the data in the file. Version 4 of the netcdf library stores data in HDF5 format files; earlier versions stored data in a custom format. The R package ncdf4 can read either format.

NetCDF files contain one or more variables, which are usually structured as regular N-dimensional arrays. For example, you might have a variable named "Temperature" that is a function of longitude, latitude, and height. NetCDF files also contain dimensions, which describe the extent of the variables' arrays. In our Temperature example, the dimensions are "longitude", "latitude", and "height". Data can be read from or written to variables in arbitrary hyperslabs (for example, you can read or write all the Temperature values at a given height, or at a given latitude).

The R package 'ncdf4' allows reading from, writing to, and creation of netCDF files, either netCDF version 3 or (optionally) netCDF version 4. If you choose to create version 4 output files, be aware that older netcdf software might only be able to read version 3 files.

Note that both the netCDF library and the HDF5 library must already be installed on your machine for this R interface to the library to work.

If you are new to netCDF files, they can be a little overwhelming, so here is a brief sketch of what documentation you need to read next.

If you want to READ data from an already-existing netCDF file, first call nc_open to open the file, then call ncvar_get to read the data from a variable in the file.

If you want to WRITE data to a new netCDF file, the procedure is to first define the dimensions your data array has, then define the variable, then create the file. So, first call ncdim_def to define the dimensions that your data exists along (for example, latitude, longitude, and time). Then call ncvar_def to define a variable that uses those dimensions, and will hold your data. Then call nc_create to create the netCDF file. Finally, call ncvar_put to write your data to the newly created netCDF file, and nc_close when you are done.

Not all features of netcdf-4 are supported yet. This version supports compression, chunking, groups, vlen strings, and multiple unlimited dimensions. User-defined types and vlen data arrays (variable-length arrays) are not supported yet.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

print.ncdf4, nc_open, nc_close, nc_create, ncdim_def, ncvar_def, ncvar_get, ncvar_put, ncvar_change_missval, ncatt_get, ncatt_put, nc_sync, nc_redef.

---

ncatt_get *Get attribute from netCDF file*

Description

Reads an attribute from a netCDF file.

Usage

```
ncatt_get( nc, varid, attname=NA, verbose=FALSE )
```

Arguments

nc	An object of class ncdf4 (as returned from nc_open), indicating what file to read from.
varid	The variable whose attribute is to be read. Can be a character string with the variable's name or an object of class ncvar4. As a special case, if varid==0, then a global (file) attribute will be read rather than a particular variable's attribute. In netcdf version 4 files, attributes can be stored in a group without an associated variable (as if they were global attributes for the group instead of for the file). In this case, set varid to a string holding the fully qualified group name using forward slashes for subgroups. For example, "group1/metadata".
attname	Name of the attribute to read; if not specified, a list containg ALL attributes of the selected variable or file is returned.
verbose	If TRUE, then debugging information is printed.

Details

This function gets an attribute from a netCDF variable (or a global attribute from a netCDF file, if the passed argument "varid" is zero). Multiple attributes are returned in a vector.

Value

If an attribute name is supplied (i.e., argument attname is given), this returns a list with two components, "hasatt" and "value". "hasatt" is TRUE if the named attribute was found, and FALSE otherwise. "value" is the (possibly vector) value of the attribute. If the on-disk type of the attribute is short or integer, then an integer value is returned. If the on-disk type is float or double, than a double value is returned. If the on-disk type is character, than a character string is returned.

If no attribute name is supplied, then this returns a list containing ALL the attributes for the specified variable along with their associated values. For example, if attlist is the list returned by this call, then names(attlist) shows all the attributes defined for the variable, and attlist[[N]] is the value of the N'th attribute.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncatt_put.

Examples

```
## Not run:
# Make a simple netCDF file
filename <- "atttest_types.nc"
```

```
dim <- ncdim_def( "X", "inches", 1:12 )
var <- ncvar_def( "Data", "unitless", dim, -1 )
ncnew <- nc_create( filename, var )

# Define some attributes of various types
attvaldbl <- 3.1415926536
ncatt_put( ncnew, var, "testatt_dbl", attvaldbl, prec="double" )
attvalfloat <- c(1.0,4.0,9.0,16.0)
ncatt_put( ncnew, var, "testatt_float", attvalfloat )
# varid=0 means it is a global attribute
ncatt_put( ncnew, 0, "globalatt_int", 32000, prec="int" )
ncatt_put( ncnew, 0, "globalatt_short", 7, prec="short" )
ncatt_put( ncnew, 0, "description",
 "this is a test file with attributes of various types")
nc_close(ncnew)

# Now illustrate the use of the ncatt_get function by reading them back in
doitfor <- function( nc, var, attname ) {
 av <- ncatt_get( nc, var, attname )
 if( av$hasatt ) {
  print(paste("File",nc$filename,", var",var,"DOES have attribute",
    attname))
  print(paste("Storage mode:",storage.mode(av$value)))
  print("Attribute value:")
  print(av$value)
 } else {
  print(paste("File",nc$filename,", var",var,"does NOT have",
   "attribute", attname))
  }
}

nc <- nc_open( filename )
var <- "Data"
doitfor( nc, var, "testatt_dbl" )
doitfor( nc, var, "testatt_float" )
doitfor( nc, var, "testatt_wacko" )
doitfor( nc, 0,   "globalatt_int" )
doitfor( nc, 0,   "globalatt_short" )
doitfor( nc, 0,   "description" )
nc_close( nc )

# Clean up after our test
file.remove( filename )


## End(Not run)
```

---

ncatt_put *Put an attribute into a netCDF file*

Description

Writes an attribute to a netCDF file.

Usage

```
ncatt_put( nc, varid, attname, attval, prec=NA, verbose=FALSE,
 definemode=FALSE )
```

Arguments

nc	An object of class ncdf4 (as returned from nc_open), indicating what file to write to.
varid	The variable whose attribute is to be written. Can be a character string with the variable's name, an object of class ncvar4, or an id contained in the "id" field of a ncvar object. As a special case, if varid==0, then a global attribute is written instead of a variable's attribute.
attname	Name of the attribute to write.
attval	Attribute to write.
prec	Precision to write the attribute. If not specified, the written precision is the same as the variable whose attribute this is. This can be overridden by specifying this argument with a value of "short", "float", "double", or "text".
verbose	Can be set to TRUE if additional information is desired while the attribute is being created.
definemode	If FALSE (the default), it is assumed that the file is NOT already in define mode. Since the file must be in define mode for this call to work, the file will be put in define mode, the attribute defined, and then the file taken out of define mode. If this argument is set to TRUE, it is assumed the file is already in define mode, and the file is also left in define mode. If you don't know what any of this means, just leave this at the default value.

Details

This function write an attribute to a netCDF variable (or a global attribute to a netCDF file, if the passed argument "varid" is zero). The type of the written variable can be controlled by the "prec" argument, if the default behavior (the precision follows that of the associated variable) is not wanted.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncatt_get.

Examples

```
## Not run:
# Make a simple netCDF file
filename <- "atttest_types.nc"
dim <- ncdim_def( "X", "inches", 1:12 )
var <- ncvar_def( "Data", "unitless", dim, -1 )
ncnew <- nc_create( filename, var )

# Define some attributes of various types
attvaldbl <- 3.1415926536
ncatt_put( ncnew, var, "testatt_dbl", attvaldbl, prec="double" )
attvalfloat <- c(1.0,4.0,9.0,16.0)
ncatt_put( ncnew, var, "testatt_float", attvalfloat )
# varid=0 means it is a global attribute
ncatt_put( ncnew, 0, "globalatt_int", 32000, prec="int" )
ncatt_put( ncnew, 0, "globalatt_short", 7, prec="short" )
ncatt_put( ncnew, 0, "description",
 "this is a test file with attributes of various types")
nc_close(ncnew)

# Clean up our test
file.remove( filename )

## End(Not run)
```

`ncdim_def` *Define a netCDF Dimension*

Description

Defines a netCDF dimension. This dimension initially only exists in memory. The dimension is later added to a netCDF variable using a call to `ncvar_def()`, and written to disk using `nc_create()`.

Usage

```
ncdim_def( name, units, vals, unlim=FALSE,
 create_dimvar=TRUE, calendar=NA, longname=name )
```

Arguments

`name`	Name of the dimension to be created (character string). The dimension name can optionally have forward slashes in it, in which case the dimension will be defined in the indicated group. For example, a dimension named `model3/run1/Longitude` will define a group named `model3`, with a subgroup named `run1`, which will hold a dimension named `Longitude`. Using groups forces a netcdf version 4 file to be written. Note that older software might not be able to read netcdf version 4 files.
`units`	The dimension's units (character string).

vals	The dimension's values (vector of numeric type). If integers are passed, the associated dimensional variable will be integer type; otherwise, it will be double precision.
unlim	If TRUE, this dimension is unlimited. Unlimited dimensions are convenient for storing, for example, data that extends over time; the time dimension can be made unlimited, and extended as needed. Or, an unlimited dimension could be the number of stations, and extended as more stations come on-line. Note that in netCDF version 4, multiple dimensions can be unlimited. In netCDF version 3, there could only be one unlimited dimension, typically the time dimension.
create_dimvar	If TRUE, a dimensional variable (aka coordinate variable) will be created for this dimension. Note: if this is set to FALSE, then 'units' must be an empty string. It is good practice to always leave this as TRUE.
calendar	If set, the specified string will be added as an attribute named "calendar" to the dimension variable. Used almost exclusively with unlimited time dimensions. Useful values include "standard" (or "gregorian"), "noleap" (or "365_day"), and "360_day").
longname	If set, AND create_dimvar is TRUE, then the created dimvar will have a long_name attribute with this value.

Details

This routine creates a netCDF dimension in memory. The created dimension can then later be passed to the routine `ncvar_def()` when defining a variable.

Note that this interface to the netCDF library by default includes that more than the minimum required by the netCDF standard. I.e., the netCDF standard allows dimensions with no units or values. This call encourages creating dimensions that have units and values, as it is useful to ensure that all dimensions have units and values, and considerably easier to include them in this call than it is to add them later. The units and values are implemented through "dimensional variables," which are variables with the same name as the dimension. By default, these dimensional variables are created automatically – there is no need for the user to create them explicitly. Dimensional variables are standard practice in netCDF files. To suppress the creation of the dimensional variable for the dimension, set passed parameter create_dimvar to FALSE. As a check, if create_dimvar is FALSE, you must ALSO pass an empty string ('') as the unit, and the values must be simple integers from 1 to the length of the dimension (e.g., 1:10 to make a dimension of length 10). This empahsizes that without a dimensional variable, a netCDF file cannot store a dimension's units or values.

The dimensional variable is usually created as a double precision floating point. The other possibility is to pass integer values (using `as.integer`, for example), in which case the dimensional variable with be integer.

The return value of this function is an object of class `ncdim4`, which describes the newly created dimension. The `ncdim` object is used for more than just creating a new dimension, however. When opening an existing file, function `nc_open` returns a `ncdf4` class object, which itself has a list of `ncdim` objects that describe all the dimensions in that existing file.

The `ncdim` object has the following fields, which are all read only: 1) name, which is a character string containing the name of the dimension; 2) units, which is a character string containing the units for the dimension, if there are any (technically speaking, this is the "units" attribute of the

associated coordinate variable); 3) vals, which is a vector containing the dimension's values (i.e., the values of the associated coordinate variable, or, if there is none, an integer sequence from 1 to the length of the dimension); 3) len, which is the length of this dimension; 4) unlim, which is a boolean indicating whether or not this is an unlimited dimension; 5) (optional) calendar, which is set if and only if the on-disk dimvar had an attribute named "calendar" (in which case, it is set to the value of that attribute).

Value

An object of class `ncdim4` that can later be passed to `ncvar_def()`.

Note

It is good practice, but not necessary, to pass the dimension's values to this routine when the dimension is created. It is also possible to write them later with a call to 'ncvar_put', using as the dimension name as the 'varid' in the call. This is useful when creating large variables with long unlimited dimensions; it can take a long time to write out the unlimited dimension's values. In this case, it can be more efficient to step through the file, writing one timestep at a time, and write that timestep's dimensional value at the same time.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncvar_def, nc_create

Examples

```
## Not run:
# Define some straightforward dimensions
x <- ncdim_def( "Lon", "degreesE", 0.5:359.5)
y <- ncdim_def( "Lat", "degreesN", as.double(-89:89))
t <- ncdim_def( "Time", "days since 1900-01-01", 1:10, unlim=TRUE)

# Make a variable with those dimensions.  Note order: time is LAST
salinity <- ncvar_def("Salinity",    "ppt",  list(x,y,t), 1.e30 )

# Create a netCDF file with this variable
ncnew <- nc_create( "salinity.nc", salinity )

nc_close(ncnew)

# Now, illustrate some manipulations of the ncdim object.
filename <- "salinity.nc"
nc <- nc_open( filename )
```

```
print(paste("File",filename,"contains",nc$ndims,"dimensions"))
for( i in 1:nc$ndims ) {
 print(paste("Here is information about dimension number",i,":"))
 d <- nc$dim[[i]]
 print(paste("    Name  :",d$name))
 print(paste("    Units :",d$units))
 print(paste("    Length:",d$len))
 print("    Values:")
 print(d$vals)
 print(paste("    Unlimited:",d$unlim))
 }
nc_close( nc )

# Clean up example
file.remove( filename )

## End(Not run)
```

---

ncvar_add *Add New netCDF Variable to Existing File*

Description

Special purpose routine for adding a new variable to a netCDF file that already exists on disk.

Usage

```
ncvar_add( nc, v, verbose=FALSE, indefine=FALSE )
```

Arguments

nc	The already-existing netCDF file we want to add a new variable to. This must be a value of class "ncdf4" returned by a call to nc_open(...,write=TRUE).
v	The variable to be added to the file. This must be a value of class "ncvar4" returned by a call to ncvar_def.
verbose	If true, prints diagnostic messages.
indefine	If true, the file is assumed to already be in define mode.

Details

There are two cases in which you might want to add a variable to a netCDF file. The first, and most common way, is when you are creating a new netCDF file. Usually when you create a netCDF file, you specify what variables you want the file to contain. This is the method most users will use to make netCDF files. To do this, do NOT use this routine; instead, pass a list of the variables you wish to have created in the output file to routine nc_create.

The second, less common, case is when you already have an existing netCDF file on disk and wish to add a new variable to it. In that case, use this routine. First define the variable you want to add to the

existing file using routine ncvar_def; then add it to the already-existing and opened (for writing) netCDF file using this routine. (This routine automatically creates any additional dimensions that are needed in the output file to handle the new variable.)

NOTE that the return value of this routine should replace the old netCDF file handle that you were using. This newly returned value reflects the modifications to the file that were accomplished by calling this routine.

Value

A handle to the netCDF file that describes the newly modified file. This is an object of type 'ncdf', the same as returned by nc_open or nc_create.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncdim_def, nc_create, ncvar_def.

Examples

```
## Not run:
#=========================================================================
# PART 1.  MAKE A TEST NETCDF FILE THAT WE WILL ADD A VARIABLE TO IN PART 2.
#=========================================================================

#----------------
# Make dimensions
#----------------
xvals <- 1:360
yvals <- -90:90

nx <- length(xvals)
ny <- length(yvals)

xdim <- ncdim_def( 'Lon', 'degreesE', xvals )
ydim <- ncdim_def( 'Lat', 'degreesE', yvals )
tdim <- ncdim_def( 'Time', 'days since 1900-01-01', 0, unlim=TRUE )

#---------
# Make var
#---------
mv <- 1.e30     # missing value
var_temp <- ncvar_def( 'Temperature', 'K', list(xdim,ydim,tdim), mv )

#---------------------
```

```
# Make new output file
#--------------------
output_fname <- 'test_real3d.nc'
ncid_new <- nc_create( output_fname, list(var_temp))

#------------------------------
# Put some test data in the file
#------------------------------
data_temp <- array(0.,dim=c(nx,ny,1))
for( j in 1:ny )
for( i in 1:nx )
        data_temp[i,j,1] <- sin(i/10)*sin(j/10)

ncvar_put( ncid_new, var_temp, data_temp, start=c(1,1,1), count=c(nx,ny,1))

#--------------------------
# Close our new output file
#--------------------------
nc_close( ncid_new )

#============================================================================
# PART 2.  ADD A NEW VARIABLE TO THE FILE
#============================================================================

#---------------------------------------------------
# Open the existing file we're going to add a var to
#---------------------------------------------------
ncid_old <- nc_open( output_fname, write=TRUE )

#-------------------------------------------------------------------
# Make a NEW variable to put into the file.  Have this new variable
# use the same dimensions already in the file
#-------------------------------------------------------------------
xdim2 <- ncid_old$dim[['Lon']]
ydim2 <- ncid_old$dim[['Lat']]
tdim2 <- ncid_old$dim[['Time']]
mv2 <- 1.e30
var_q <- ncvar_def( 'Humidity', 'g/kg', list(xdim2,ydim2,tdim2), mv2 )

ncid_old <- ncvar_add( ncid_old, var_q ) # NOTE this returns a modified netcdf file handle

#-------------------------------------------------------------------
# Make a DIFFERENT new var that will be added to the file.  This var
# uses a dim that does NOT already exist in the file.
#-------------------------------------------------------------------
zdim   <- ncdim_def( 'Level', 'hPa', seq(1000,100,by=-100))
var_cf <- ncvar_def( 'CloudFraction', 'percent', list(xdim2,ydim2,zdim,tdim2), mv2 )

ncid_old <- ncvar_add( ncid_old, var_cf )

print(ncid_old)

nc_close( ncid_old )
```

```
# Clean up our example
file.remove( output_fname )

## End(Not run)
```

`ncvar_change_missval` *Change the Missing Value For a netCDF Variable*

Description

Changes the missing_value attribute for a netCDF variable.

Usage

```
ncvar_change_missval( nc, varid, missval )
```

Arguments

`nc`	An object of class `ncdf4`, as returned by `nc_open(...,write=TRUE)` or `nc_create`.
`varid`	Either the name of the variable or an `ncvar` object indicating whose missing value will be changed.
`missval`	The missing value to change to.

Details

Note: this specialty function is only used to change a variable's missing value after it has already been defined, which is rare. The proper way to set a variable's missing value in the first place is by setting the missing value argument to routine `ncvar_def` appropriately.

Missing values are special values in netCDF files whose value is to be taken as indicating the data is "missing". This is a convention, and is indicated by the netCDF variable having an attribute named "missing_value" that holds this number. This function sets the "missing_value" attribute for a variable.

R uses a similar concept to indicate missing values, the "NA" value. When the ncdf library reads in data set from a pre-existing file, all data values that equal that variable's missing value attribute appear to the R code as being "NA" values. When the R code writes values to a netCDF variable, any "NA" values are set to that variable's missing value before being written out. This makes the mapping between netCDF's "missing_value" attribute and R's "NA" values transparent to the user.

For this to work, though, the user still has to specify a missing value for a variable. Usually this is specified when the variable is created, as a required argument to `ncvar_def`. However, sometimes it is useful to add (or change) a missing value for variable that already exists in a disk file. This function enables that.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncvar_def.

Examples

```
## Not run:
# Make an example netCDF file with a given missing value.  We will
# then change the missing value in the file using ncvar_change_missval

origMissVal <- -1.
dimX   <- ncdim_def( "X", "meters", 1:7 )
varAlt <- ncvar_def( "Altitude", "km", dimX, origMissVal )
ncnew  <- nc_create( "transect.nc", varAlt )
data <- c(10.,2.,NA,1.,7.,NA,8.)
ncvar_put( ncnew, varAlt, data )
nc_close(ncnew)

# At this point, the actual data values in the netCDF
# file will be: 10 2 -1 1 7 -1 8
# because the "NA" values were filled with the missing
# value, -1.  Also, the missing_value attribute of variable
# "varAlt" will be equal to -1.

# Now change the missing value to something else.  Remember
# we have to open the file as writable to be able to change
# the missing value on disk!

newMissVal <- 999.9
nc <- nc_open( "transect.nc", write=TRUE )
varname <- "Altitude"
data <- ncvar_get( nc, varname )  # data now has: 10., 2., NA, 1., 7., NA, 8.
print(data)
ncvar_change_missval( nc, varname, newMissVal )
ncvar_put( nc, varname, data )
nc_close(nc)

# Now, the actual data values in the netCDF file will be:
# 10 2 999.9 1 7 999.9 8
# and the variables "missing_value" attributre will be 999.9

# **NOTE** that we had to explicitly read in the data and write
# it out again in order for the on-disk missing values in the
# data array to change!  The on-disk missing_value attribute for
# the variable is set automatically by this function, but it is
# up to you whether or not you want to read in all the existing
# data and change the values to the new missing value.

# Clean up our example
```

```
file.remove( "transect.nc" )

## End(Not run)
```

---

ncvar_def *Define a netCDF Variable*

Description

Defines a netCDF variable. This variable initially only exists in memory. It is later written to disk using `nc_create()`.

Usage

```
ncvar_def( name, units, dim, missval=NULL, longname=name, prec="float",
shuffle=FALSE, compression=NA, chunksizes=NA, verbose=FALSE )
```

Arguments

`name`	Name of the variable to be created (character string). The name can optionally have forward slashes in it, in which case the variable will be defined in the indicated group. For example, a variable named `model3/run1/Temperature` will define a group named `model3`, with a subgroup named `run1`, which will hold a variable named `Temperature`. Using groups forces a netcdf version 4 file to be written. Note that older software might not be able to read netcdf version 4 files.
`units`	The variable's units (character string). Or, pass a zero length string (") to have no units attribute.
`dim`	The variable's dimension(s) (one or a list of "ncdim4" class objects, as returned by `ncdim_def`). To create a variable with NO dimensions, pass an empty list ("list()").
`missval`	The variable's missing value. If NO missing value is desired, pass a NULL, or omit this argument entirely. If a NaN missing value is desired, pass an NA.
`longname`	Optional longer name for the variable, which is assigned to the variable's "long_name" attribute. For example, a variable named "TS" might have the longname "Surface Temperature"
`prec`	Precision of the created variable. Valid options: 'short' 'integer' 'float' 'double' 'char' 'byte'. See the special note below for how to create a character variable (strings).
`shuffle`	Turns on (if TRUE) or off (if FALSE, the default) the shuffle filter. According to netcdf docs, turning the shuffle filter on can improve compression for integer variables. Turning the shuffle filter on forces the created file to be in netcdf version 4 format, which will not be compatible with older software that only reads netcdf version 3 files.

compression	If set to an integer between 1 (least compression) and 9 (most compression), this enables compression for the variable as it is written to the file. Turning compression on forces the created file to be in netcdf version 4 format, which will not be compatible with older software that only reads netcdf version 3 files.
chunksizes	If set, this must be a vector of integers with a length equal to the number of dimensions in the variable. When data from this variable is written to the file, it will be buffered in blocks as indicated by the chunksize. The order of dimensions in this vector is the standard R ordering of XYZT. In some instances, setting a chunksize that reflects how the variable's data will be read or written can greatly reduce read or write times. See the netcdf documentation for more detail on how to set this parameter. Enabling this feature forces the created file to be in netcdf version 4 format, which will not be compatible with older software that only reads netcdf version 3 files.
verbose	Print debugging information.

Details

This routine creates a netCDF variable in memory. The variable can then be passed to the routine `nc_create` when writing a file to disk.

Note that this interface to the netCDF library includes more than the minimum required by the netCDF standard. I.e., the netCDF standard allows variables with no units or missing values. This call requires units and a missing value, as it is useful to ensure that all variables have units and missing values, and considerably easier to include them in this call than it is to add them later. The units and missing value are implemented through attributes to the variable, named "units" and "missing_value", respectively. This is standard practice in netCDF files.

After a variable is defined with this call, and created on disk using `nc_create`, then data values for the variable can be written to disk using `ncvar_put`.

This function returns a `ncvar` object, which describes the newly-created variable. However, the `ncvar` object is used for more than just creating new variables. The function `nc_open` returns a `ncdf4` class object that itself contains a list of `ncvar4` objects that describe the variables in an existing, on-disk netCDF file. (Note that coordinate variables are NOT included in this list. Attributes of the coordinate variables are kept in the `ncdim4` class object instead.)

The `ncvar4` class object has the following fields, which are all read-only: 1) name, which is a character string containing the name of the variable; 2) units, which is a character string containing the contents of the variable's "units" attribute; 3) missval, which contains the contents of the variable's "missing_value" attribute; 4) longname, which is the contents of the variable's "long_name" attribute, or defaults to the name of the variable if there is no "long_name" attribute; 5) ndims, which is the number of dimensions this variable has; 6) dim, which is a list of objects of class "ncdim4" (see `ncdim_def`), and describe this variable's dimensions; 7) unlim, which is TRUE if this variable has an unlimited dimension and FALSE otherwise; 8) varsize, which is a convenience array that gives the shape of the variable (in XYZT ordering).

Note that the missval attribute does not need to be used much in R, because R's special value NA is fully supported. I.e., when data is read in from an existing file, any values equal to the "missing" value are set to NA. When data is written out, any NAs are set equal to the missing value. If not explicitly set by the user, a default value of 1.e30 is used for the missing value.

CHARACTER VARIABLES: Character-type variables (i.e., strings) are supported using the original netcdf library approach, which is used because it is backwards compatible with the older version 3 of the netcdf library. This approach uses fixed-size strings, which should be declared to be the maximum length of the string that will be stored. If the maximum string size is N, then an auxiliary dimension is first defined, with values running from 1 to N. For example, if N=12 and we call the new auxiliary dimension "nchar", then the code could look like this: dimnchar <- ncdim_def("nchar", "", 1:12, create_dimvar=FALSE). The character type variable is then defined with the first dimension being this new auxiliary variable: varcolors <- ncvar_def("colors", "", list(dimnchar, dimcolorno), prec="char"). See the manual page for ncvar_put for a worked example.

Value

An object of class ncvar4 that can later be passed to nc_create().

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncdim_def, nc_create, ncvar_put.

Examples

```
## Not run:
# Define an integer dimension
dimState <- ncdim_def( "StateNo", "count", 1:50 )

# Make an integer variable.  Note that an integer variable can have
# a double precision dimension, or vice versa; there is no fixed
# relationship between the precision of the dimension and that of the
# associated variable.  We just make an integer variable here for
# illustration purposes.
varPop <- ncvar_def("Pop", "count", dimState, -1,
  longname="Population", prec="integer")

# Create a netCDF file with this variable
ncnew <- nc_create( "states_population.nc", varPop )

# Write some values to this variable on disk.
popAlabama <- 4447100
ncvar_put( ncnew, varPop, popAlabama, start=1, count=1 )

# Add source info metadata to file
ncatt_put( ncnew, 0, "source", "Census 2000 from census bureau web site")
```

```
nc_close(ncnew)

# Now illustrate some manipulations of the var.ncdf object
filename <- "states_population.nc"
nc <- nc_open(filename)
print(paste("File",nc$filename,"contains",nc$nvars,"variables"))
for( i in 1:nc$nvars ) {
 v <- nc$var[[i]]
 print(paste("Here is information on variable number",i))
 print(paste("   Name: ",v$name))
 print(paste("   Units:",v$units))
 print(paste("   Missing value:",v$missval))
 print(paste("   # dimensions :",v$ndims))
 print(paste("   Variable size:",v$varsize))
 }

# Illustrate creating variables of various types.  You will find
# that the type of the missing_value attribute automatically follows
# the type of the variable.
dimt <- ncdim_def( "Time", "days", 1:3 )
missval <- -1
varShort <- ncvar_def( "varShort", "meters", dimt, missval, prec="short")
varInt   <- ncvar_def( "varInt",   "meters", dimt, missval, prec="integer")
varFloat <- ncvar_def( "varFloat", "meters", dimt, missval, prec="single")
varDouble<- ncvar_def( "varDouble","meters", dimt, missval, prec="double")
nctypes <- nc_create("vartypes.nc", list(varShort,varInt,varFloat,varDouble) )
nc_close(nctypes)

# Clean up example
file.remove( filename )
file.remove( "vartypes.nc" )

## End(Not run)
```

---

ncvar_get *Read data from a netCDF file*

Description

Reads data from an existing netCDF file.

Usage

```
ncvar_get(nc, varid=NA, start=NA, count=NA, verbose=FALSE,
signedbyte=TRUE, collapse_degen=TRUE, raw_datavals=FALSE )
```

Arguments

nc	An object of class ncdf4 (as returned by either function nc_open or function nc_create), indicating what file to read from.

varid	What variable to read the data from. Can be a string with the name of the variable or an object of class ncvar4 If left unspecified, the function will determine if there is only one variable in the file and, if so, read from that. If left unspecified and there are multiple variables in the file, an error is generated. This argument can also, optionally, specify the name of a dimension (usually the unlimited dimension) in order to read values from a coordinate variable. Note this is not usual practice, because the ncdim object already contains all the dimension's values in the field named "vals". However, it can sometimes be faster to turn off this automatic reading of the unlimited dimension's values by using nc_open(filename, readunlim=FALSE), then read the dimension values in later with this function.
start	A vector of indices indicating where to start reading the passed values (beginning at 1). The length of this vector must equal the number of dimensions the variable has. Order is X-Y-Z-T (i.e., the time dimension is last). If not specified, reading starts at the beginning of the file (1,1,1,...).
count	A vector of integers indicating the count of values to read along each dimension (order is X-Y-Z-T). The length of this vector must equal the number of dimensions the variable has. If not specified and the variable does NOT have an unlimited dimension, the entire variable is read. As a special case, the value "-1" indicates that all entries along that dimension should be read.
verbose	If TRUE, then progress information is printed.
signedbyte	If TRUE (default), then on-disk byte variables are interpreted as signed. This is in accord with the netCDF standard. If FALSE, then on-disk byte variables are interpreted as unsigned.
collapse_degen	If TRUE (the default), then degenerate (length==1) dimensions in the returned array are removed.
raw_datavals	If TRUE, then the actual raw data values from the file are returned with no conversion to NA (if equal to the missing value/fill value) or scale/offset applied. Default is FALSE.

Details

This routine reads data values from a variable in an existing netCDF file. The file must already have been opened with a call to nc_open.

Returned values will be in ordinary R double precision if the netCDF variable type is float or double. Returned values will be in R's integer storage mode if the netCDF variable type is short or int. Returned values will be of character type if the netCDF variable is of character type.

Values of "NA" are supported; values in the data file that match the variable's missing value attribute are automatically converted to "NA" before being returned to the user. See ncvar_change_missval for more information.

Data in a netCDF file is conceived as being a multi-dimensional array. The number and length of dimensions is determined when the variable is created. The 'start' and 'count' indices that this routine takes indicate where the writing starts along each dimension, and the count of values along each dimension to write. Note that the special count value "-1" means "all the values along that dimension".

If the variable in the netCDF file has a scale and/or offset attribute defined, the returned data are automatically and silently scaled and/or offset as requested.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncvar_put.

Examples

```
## Not run:
# Start with the simplest example.  If the file only has one variable in it,
# you can read the data as easily as this:
#
nc <- nc_open("salinity.nc")
# NOTE how not specifying varid reads the "only" var in the file
data <- ncvar_get( nc )
nc_close(nc)

# In this next example we read values from file "writevals.nc", which is created by
# the R code in the example section for function "ncvar_put".  We open the
# file with readunlim=FALSE for potentially faster access, and to illustrate
# (below) how to read in the unlimited dimension values.
#
nc <- nc_open( "writevals.nc", readunlim=FALSE )

print(paste("The file has",nc$nvars,"variables"))

# This illustrates how to read all the data from a variable
v1 <- nc$var[[1]]
data1 <- ncvar_get( nc, v1 ) # by default, reads ALL the data
print(paste("Data for var ",v1$name,":",sep=""))
print(data1)

# This shows how the shape of the read data is preserved
v2 <- nc$var[[2]]
data2 <- ncvar_get( nc, v2 )
print(paste("Var 2 has name",v2$name,"and is of shape",dim(data2),
 ". Here are the values:"))
print(data2)

# This illustrates how to read data one timestep at a time.  In this
# example we will elaborately show how to deal with a variable whose
# shape is completely unknown (i.e., how many dimensions, and what their
# sizes are).  We will also, for illustration of a common case, show how
# to read in the values of the time dimension at each timestep.
v3      <- nc$var[[3]]
varsize <- v3$varsize
ndims   <- v3$ndims
```

```
nt      <- varsize[ndims]  # Remember timelike dim is always the LAST dimension!
for( i in 1:nt ) {
 # Initialize start and count to read one timestep of the variable.
 start <- rep(1,ndims) # begin with start=(1,1,1,...,1)
 start[ndims] <- i # change to start=(1,1,1,...,i) to read timestep i
 count <- varsize # begin w/count=(nx,ny,nz,...,nt), reads entire var
 count[ndims] <- 1 # change to count=(nx,ny,nz,...,1) to read 1 tstep
 data3 <- ncvar_get( nc, v3, start=start, count=count )

 # Now read in the value of the timelike dimension
 timeval <- ncvar_get( nc, v3$dim[[ndims]]$name, start=i, count=1 )

 print(paste("Data for variable",v3$name,"at timestep",i,
  " (time value=",timeval,v3$dim[[ndims]]$units,"):"))
 print(data3)
 }

nc_close(nc)

## End(Not run)
```

---

ncvar_put *Write data to a netCDF file*

Description

Writes data to an existing netCDF file. The variable to be written to must already exist on disk (i.e., you must call either nc_create or nc_open before calling this function).

Usage

```
ncvar_put( nc, varid, vals, start=NA, count=NA, verbose=FALSE, na_replace="fast" )
```

Arguments

nc	An object of class ncdf4 (as returned by either function nc_open or nc_create), indicating what file to write to.
varid	What variable to write the data to. Can be a string with the name of the variable or an object of class ncvar4, as returned by ncvar_def or nc_open.
vals	The values to be written.
start	A vector of indices indicating where to start writing the passed values (starting at 1). The length of this vector must equal the number of dimensions the variable has. Order is X-Y-Z-T (i.e., the time dimension is last). If not specified, writing starts at the beginning of the file (1,1,1,...).
count	A vector of integers indicating the count of values to write along each dimension (order is X-Y-Z-T). The length of this vector must equal the number of dimensions the variable has. If not specified and the variable does NOT have

an unlimited dimension, the entire variable is written. If the variable has an unlimited dimension, this argument must be specified. As a special case, the value "-1" indicates that all entries along that dimension should be written.

`verbose` If true, prints information while executing.

`na_replace` This is either the string "fast" or the string "safe". When the 'vals' array is written out, NA's have to be replaced by the missing value specified when the variable was created. If na_replace is "fast", then this is done in-place without copying the 'vals' data array. This results in the passed 'vals' array being modified such that NA's are replaced with the missing value. This is fast but not standard in R and may be unexpected. If na_replace is "safe" then the vals array is copied before the NA replacement, so that the vals array is not modified. This is more expected and standard R, but can be slow and might cause memory issues if a very large 'vals' array is passed in. Default value is "fast".

Details

This routine writes data values to a variable in a netCDF file. The file should have either been created with `nc_create`, or opened with `nc_open` called with parameter `write=TRUE`.

Note that the data type (i.e., precision) of the values written to the file is determined when the variable is created; in particular, it does not matter what type you pass to this function to be written. In other words, if the variable was created with type 'integer', passing double precision values to this routine will still result in integer values being written to disk.

Values of "NA" are supported; they are converted to the netCDF variable's missing value attribute before being written. See `ncvar_change_missval` for more information.

Data in a netCDF file is conceived as being a multi-dimensional array. The number and length of dimensions is determined when the variable is created. The 'start' and 'count' indices that this routine takes indicate where the writing starts along each dimension, and the count of values along each dimension to write.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

`ncdim_def`, `nc_create`, `nc_open`, `ncvar_get`.

Examples

```
## Not run:
#--------------------------------
# Make a few dimensions we can use
#--------------------------------
nx <- 3
ny <- 4
```

```
nt <- 5
xvals <- (1:nx)*100.
dimX <- ncdim_def( "X", "meters", xvals )
dimY <- ncdim_def( "Y", "meters", (1:ny)*100. )
dimT <- ncdim_def( "Time", "seconds", (1:nt)/100., unlim=TRUE )

#-----------------------------------------------------------------
# Make varables of various dimensionality, for illustration purposes
#-----------------------------------------------------------------
mv <- 1.e30  # missing value to use
var1d <- ncvar_def( "var1d", "units", dimX, mv )
var2d <- ncvar_def( "var2d", "units", list(dimX,dimY), mv )
var3d <- ncvar_def( "var3d", "units", list(dimX,dimY,dimT), mv )

#--------------------
# Create the test file
#--------------------
nc <- nc_create( "writevals.nc", list(var1d,var2d,var3d) )

#----------------------------
# Write some data to the file
#----------------------------
data1d <- runif(nx)
ncvar_put( nc, var1d, data1d ) # no start or count: write all values
ncvar_put( nc, var1d, 27.5, start=3, count=1 ) # Write a value to the third slot

data2d <- runif(nx*ny)
ncvar_put( nc, var2d, data2d ) # no start or count: write all values

#--------------------------------
# Write a 1-d slice to the 2d var
#--------------------------------
ncvar_put( nc, var2d, data1d, start=c(1,2), count=c(nx,1) )

#---------------------------------------------------------------
# Note how "-1" in the count means "the whole dimension length",
# which equals nx in this case
#---------------------------------------------------------------
ncvar_put( nc, var2d, data1d, start=c(1,3), count=c(-1,1) )

#-----------------------------------------------------------------------------
# The 3-d variable has an unlimited dimension.  We will loop over the timesteps,
# writing one 2-d slice per timestep.
#-----------------------------------------------------------------------------
for( i in 1:nt)
 ncvar_put( nc, var3d, data2d, start=c(1,1,i), count=c(-1,-1,1) )

nc_close(nc)

#---------------------------------------------------------------------
# Illustrate creating a character type variable
#---------------------------------------------------------------------
cnames   <- c("red", "orange", "green", "yellow", "puce", "colorwithverylongname" )
```

```
nstrings <- length(cnames)

#-----------------------------------------------------------
# Make dimensions. Setting "dimnchar" to have a length of 12
# means that the maximum color name
# length can be 12.  Longer names will be truncated to this.
# We don't need dimvars for this example.
#-----------------------------------------------------------
dimnchar   <- ncdim_def("nchar",   "", 1:12, create_dimvar=FALSE )
dimcolorno <- ncdim_def("colorno", "", 1:nstrings, create_dimvar=FALSE )

#---------------------------------------------------------------------
# NOTE in the following call that units is set to the empty string (""),
# which suppresses creation of a units attribute, and the missing value
# is entirely omitted, which suppresses creation of the missing value att
#---------------------------------------------------------------------
varcolors  <- ncvar_def("colors", "", list(dimnchar, dimcolorno), " ",
   prec="char" )

ncid <- nc_create( "colornames.nc", list(varcolors) )

ncvar_put( ncid, "colors", cnames, verbose=TRUE )

nc_close( ncid )

#---------
# Clean up
#---------
file.remove( "colornames.nc" )
file.remove( "writevals.nc" )

## End(Not run)
```

---

ncvar_rename *Rename an Existing Variable in a netCDF File*

Description

Renames an existing variable that currently is part of a netCDF file that is on disk.

Usage

```
ncvar_rename( nc, old_varname, new_varname, verbose=FALSE )
```

Arguments

nc	The already-existing netCDF file that we want to manipulate. This must be a value of class "ncdf4" returned by a call to nc_open(...,write=TRUE).

old_varname	The variable in the file that is to be renamed. This can be a string with the name of the variable to be renamed, or a value of class "ncvar4" returned by a call to ncvar_def().
new_varname	A string containing the new name of the variable.
verbose	If true, run verbosely.

Details

This call allows you to rename a variable that already exists in a netCDF file.

NOTE that the return value of this routine should replace the old netCDF file handle that you were using. This newly returned value reflects the modifications to the file that were accomplished by calling this routine.

Value

The updated value of nc that contains the new name. This needs to replace the old value of nc in the code. I..e, `ncid <- ncvar_rename( ncid, ... )`.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncdim_def, nc_create, ncvar_def.

Examples

```
## Not run:
#=================================================================
# PART 1.  MAKE A TEST NETCDF FILE THAT WE WILL MANIPULATE IN PART 2
#=================================================================

#----------------
# Make dimensions
#----------------
xvals <- 1:360
yvals <- -90:90

nx <- length(xvals)
ny <- length(yvals)

xdim <- ncdim_def( 'Lon', 'degreesE', xvals )
ydim <- ncdim_def( 'Lat', 'degreesE', yvals )
tdim <- ncdim_def( 'Time', 'days since 1900-01-01', 0, unlim=TRUE )
```

```
#---------
# Make var
#---------
mv <- 1.e30     # missing value
var_temp <- ncvar_def( 'Temperature', 'K', list(xdim,ydim,tdim), mv )

#---------------------
# Make new output file
#---------------------
output_fname <- 'test_real3d.nc'
ncid_new <- nc_create( output_fname, list(var_temp))

#-----------------------------
# Put some test data in the file
#-----------------------------
data_temp <- array(0.,dim=c(nx,ny,1))
for( j in 1:ny )
for( i in 1:nx )
        data_temp[i,j,1] <- sin(i/10)*sin(j/10)

ncvar_put( ncid_new, var_temp, data_temp, start=c(1,1,1), count=c(nx,ny,1))

#--------------------------
# Close our new output file
#--------------------------
nc_close( ncid_new )

#============================================================================
# PART 2.  RENAME A NEW VARIABLE TO THE FILE
#============================================================================

#-------------------------------------------------
# Open the existing file we're going to manipulate
#-------------------------------------------------
ncid_old <- nc_open( output_fname, write=TRUE )

old_varname <- 'Temperature'
new_varname <- 'T'

ncid_old <- ncvar_rename( ncid_old, old_varname, new_varname )

print(ncid_old)

nc_close( ncid_old )

# Clean up our example
file.remove( output_fname )

## End(Not run)
```

---

nc_close *Close a netCDF File*

Description

Closes an open netCDF file, which flushes any unwritten data to disk. Always close a netCDF file when you are done with it! You are risking data loss otherwise.

Usage

```
nc_close( nc )
```

Arguments

nc	An object of class ncdf4 (as returned by either function nc_open or function nc_create.

Details

Data written to a netCDF file is cached in memory, for better performance. This data is only written out to disk when the file is closed. Therefore, always remember to close a netCDF file when done with it.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

nc_sync.

Examples

```
## Not run: nc <- nc_open("salinity.nc")
## Not run: data <- ncvar_get( nc ) # Read the "only" var in the file
## Not run: nc_close(nc)
```

---

nc_create *Create a netCDF File*

Description

Creates a new netCDF file on disk, given the variables the new file is to contain.

Usage

```
nc_create( filename, vars, force_v4=FALSE, verbose=FALSE )
```

Arguments

filename	Name of the netCDF file to be created.
vars	Either an object of class ncvar4 describing the variable to be created, or a vector (or list) of such objects to be created.
force_v4	If TRUE, then the created output file will always be in netcdf-4 format (which supports more features, but cannot be read by version 3 of the netcdf library). If FALSE, then the file is created in netcdf version 3 format UNLESS the user has requested features that require version 4. Deafult is FALSE.
verbose	If TRUE, then information is printed while the file is being created.

Details

This routine creates a new netCDF file on disk. The routine must be called with the variables that will be created in the file. Keep in mind that the new file may not actually be written to disk until nc_close is called. Always call nc_close when you are done with your file, or before exiting R!

Value

An object of class ncdf4, which has the fields described in nc_open.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncdim_def, ncvar_def.

Examples

```
## Not run:
# Define an integer dimension
dimState <- ncdim_def( "StateNo", "count", 1:50 )

# Make an integer variable.  Note that an integer variable can have
# a double precision dimension, or vice versa; there is no fixed
# relationship between the precision of the dimension and that of the
# associated variable.  We just make an integer variable here for
# illustration purposes.
varPop <- ncvar_def("Pop", "count", dimState, -1,
  longname="Population", prec="integer")

# Create a netCDF file with this variable
ncnew <- nc_create( "states_population.nc", varPop )

# Write some values to this variable on disk.
```

```
popAlabama <- 4447100
ncvar_put( ncnew, varPop, popAlabama, start=1, count=1 )

nc_close(ncnew)

# Clean up example
file.remove( "states_population.nc" )

## End(Not run)
```

---

nc_enddef *Takes a netCDF file out of define mode*

Description

Changes a netCDF that is currently in define mode back into data mode.

Usage

```
nc_enddef( nc )
```

Arguments

nc	An object of class ncdf4 (as returned by either function nc_open or function nc_create, indicating what file to operate upon.

Details

NOTE: typical users will never need to use this function.

NetCDF files can be in "define mode", at which time dimensions and variables can be defined, or new attributes added to a file, or in "data mode", at which time data can be read from the file. This call puts a file that is currently in define mode back into data mode. The ncdf4 package manages this process transparently, so normally, an end user will not need to call this explicitly.

Note

The typical user will never need this call, nor will ever have to worry about "define mode" or "data mode". THIS CALL IS PROVIDED FOR ADVANCED USERS ONLY! If the user goes through this package's standard functional interface, the file will always automatically be set to whatever mode it needs to be in without the user having to do anything. For example, the call to write an attribute (ncatt_put) handles this automatically.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

nc_redef.

Examples

```
# This function is for advanced useage only, and will never
# be needed by the typical users R code.
```

---

nc_open *Open a netCDF File*

Description

Opens an existing netCDF file for reading (or, optionally, writing).

Usage

```
nc_open( filename, write=FALSE, readunlim=TRUE, verbose=FALSE,
 auto_GMT=TRUE, suppress_dimvals=FALSE, return_on_error=FALSE )
```

Arguments

filename	Name of the existing netCDF file to be opened.
write	If FALSE (default), then the file is opened read-only. If TRUE, then writing to the file is allowed.
readunlim	When invoked, this function reads in the values of all dimensions from the associated variables. This can be slow for a large file with a long unlimited dimension. If set to FALSE, the values for the unlimited dimension are not automatically read in (they can be read in later, manually, using ncvar_get()).
verbose	If TRUE, then messages are printed out during execution of this function.
auto_GMT	If TRUE, then GMT files are automatically detected. Does not yet do anything.
suppress_dimvals	If TRUE, then NO dimensional values are automatically read in from the file. (Use this if there are so many dimensional values that a out-of-memory error is generated).
return_on_error	If TRUE, then nc_open always returns, and returned list element $error will be TRUE if an error was encountered and FALSE if no error was encountered. If return_on_error is FALSE (the default), nc_open halts with an error message if an error is encountered.

Details

This routine opens an existing netCDF file for reading (or, if write=TRUE, for writing). To create a new netCDF file, use nc_create instead.

In addition to simply opening the file, information about the file and its contents is read in and stored in the returned object, which is of class ncdf4. This class has the following user-accessible fields, all of which are read-only: 1) filename, which is a character string holding the name of the file; 2) ndims, which is an integer holding the number of dimensions in the file; 3) nvars, which is an integer holding the number of the variables in the file that are NOT coordinate variables (aka dimensional variables); 4) natts, which is an integer holding the number of global attributes; 5) unlimdimid, which is an integer holding the dimension id of the unlimited dimension, or -1 if there is none; 6) dim, which is a list of objects of class ncdim4; 7) var, which is a list of objects of class ncvar4; 8) writable, which is TRUE or FALSE, depending on whether the file was opened with write=TRUE or write=FALSE.

The concept behind the R interface to a netCDF file is that the ncdf4 object returned by this function, as well as the list of ncdim4 objects contained in the ncdf object's "dim" list and the ncvar4 objects contained in the ncdf object's "var" list, completely describe the netCDF file. I.e., they hold the entire contents of the file's metadata. Therefore, there are no R interfaces to the explicit netCDF query functions, such as "nc_inq_nvars" or "nc_inq_natts". The upshot is, look in the ncdf4 object or its children to get information about the netCDF file. (Note: the ncdim4 object is described in the help file for ncdim_def; the ncvar4 object is described in the help file for ncvar_def).

Missing values: R uses "NA" as a missing value. Netcdf files have various standards for indicating a missing value. The most common is that a variable will have an attribute named "_FillValue" indicating the value that should be interpreted as a missing value. (For example, the _FillValue attribute might have the value of 1.e30, indicating that any data in the netcdf file with a value of 1.e30 should be interpreted as a missing value.) If the "_FillValue" attribute is found, then the ncdf4 package transparently maps all the netcdf file's missing values to NA's; this is the most common case. The attribute "missing_value" is also recognized if there is no "_FillValue" attribute.

Some netcdf files specify both a "_FillValue" and a "missing_value" attribute for a variable. If these two attributes have the same value, then everything is fine. If they have different values, I consider this a malformed netcdf file and I suggest you contact the person who made your netcdf file to fix it. In this event you can set the "raw_datavals" flag in the ncvar_get() call and handle the conflicting missing values however you want.

If the netcdf file does not have a missing value, then the ncdf4 package assigns a default missing value of 1.e30 to the netcdcf file so that R NA's, which are always possible in the R environment, can be sensibly handled in the netcdf file. On rare occasions this can cause problems with non-compliant or incorrect netcdf files that implicitly use some particular value, for example 9.96921e+36, to indicate a missing value but without setting a proper _FillValue attribute. The best way to fix such netcdf files is to explicitly put in the correct _FillValue attribute using an ncatt_put call.

Value

An object of class ncdf4 that has the fields described above.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

ncdim_def, ncvar_def, ncatt_put.

Examples

```
## Not run:
# Define an integer dimension
dimState <- ncdim_def( "StateNo", "count", 1:50 )

# Make an integer variable.  Note that an integer variable can have
# a double precision dimension, or vice versa; there is no fixed
# relationship between the precision of the dimension and that of the
# associated variable.  We just make an integer variable here for
# illustration purposes.
varPop <- ncvar_def("Pop", "count", dimState, -1,
 longname="Population", prec="integer")

# Create a netCDF file with this variable
ncnew <- nc_create( "states_population.nc", varPop )

# Write some values to this variable on disk.
popAlabama <- 4447100
ncvar_put( ncnew, varPop, popAlabama, start=1, count=1 )

# Add source info metadata to file
ncatt_put( ncnew, 0, "source", "Census 2000 from census bureau web site")

nc_close(ncnew)

# Now open the file and read its data
ncold <- nc_open("states_population.nc")
data <- ncvar_get(ncold)
print("here is the data in the file:")
print(data)
nc_close( ncold )

# Clean up example
file.remove( "states_population.nc" )

## End(Not run)
```

---

nc_redef *Puts a netCDF file back into define mode*

Description

Puts a netCDF that is not currently in define mode back into define mode.

Usage

```
nc_redef( nc )
```

Arguments

nc	An object of class ncdf4 (as returned by either function nc_open(..., write=TRUE) or function nc_create, indicating what file to operate on.

Details

Typically, users will never need this function.

NetCDF files can be in "define mode", at which time dimensions and variables can be defined, or new attributes added to a file, or in "data mode", at which time data can be read from the file. This call puts a file that is currently in data mode back into define mode. This functionality is handled transparently by the ncdf4 library, so users will never need to call this unless they are doing advanced manipulations of netcdf files.

Note

The typical user will never need this call, nor will ever have to worry about "define mode" or "data mode". THIS CALL IS PROVIDED FOR ADVANCED USERS ONLY! If the user goes through this package's standard functional interface, the file will always automatically be set to whatever mode it needs to be in without the user having to do anything. For example, the call to write an attribute (ncatt_put) handles this automatically.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

nc_enddef.

Examples

```
# This function is for advanced useage only, and will never
# be needed by the typical users R code.
```

nc_sync *Synchronize (flush to disk) a netCDF File*

Description

Flushes any pending operations on a netCDF file to disk.

Usage

```
nc_sync( nc )
```

Arguments

nc	An object of class ncdf4 that is opened for writing (as returned by either function nc_open(..., write=TRUE) or function nc_create, indicating what file is being written to.

Details

Data in a netCDF file is cached in memory, for better performance. An example of when this might be bad is if a long-running job writes one timestep of the output file at a time; if the job crashes near the end, the results of many timesteps might be lost. In such an event, the user can manually force any cached data to be written to disk using this call.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

Examples

```
## Not run:
# The time you would use the sync.ncdf function is when you have an unlimited
# dimension and are writing to the file timestep-by-timestep. Make a netCDF file
# that has an unlimited dimension for illustration.
nx <- 5
ny <- 8
dimx <- ncdim_def( "X", "meters", 1:nx )
dimy <- ncdim_def( "Y", "meters", 1:ny )
dimt <- ncdim_def( "Time", "days since 1900-01-01", 0, unlim=TRUE )

vartemp <- ncvar_def( "Temperature", "degC", list(dimx,dimy,dimt), 1.e30 )
nc  <- nc_create( "temperature.nc", vartemp )

nt <- 10  # Imagine this is actually some very large number of timesteps
for( i in 1:nt ) {
```

```
 # Long, slow computation to get the data ... for illustration, we just
 # use the following:
 data <- runif(nx*ny)

 # Write the data to this timestep
 ncvar_put( nc, vartemp, data, start=c(1,1,i), count=c(nx,ny,1) )

 # Write the time value for this timestep as well
 timeval <- i*10
 ncvar_put( nc, dimt, timeval, start=i, count=1 )

 # Flush this timestep's data to the file so we dont lose it
 # if there is a crash or other problem
 nc_sync( nc )
 }

# Always remember to close the file when done!!
nc_close(nc)

# Clean up example
file.remove( "temperature.nc" )

## End(Not run)
```

`nc_version` *Report version of ncdf4 library*

Description

Returns a string that is the version number of the ncdf4 package.

Usage

```
nc_version()
```

Details

Note that the returned value it is a string, not a floating point number.

Value

A string (not float) that is the version number of the ncdf4 package.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

print.ncdf4 *Print Information About a netCDF File*

Description

Prints information about a netCDF file, including the variables and dimensions it contains.

Usage

```
 ## S3 method for class 'ncdf4'
print( x, ... )
```

Arguments

x	An object of class "ncdf4".
...	Extra arguments are passed to the generic print function.

Details

NetCDF files contain variables, which themselves have dimensions. This routine prints out useful information about a netCDF file's variables and dimensions. It is overloaded on the regular print function, so if "nc" is an object of class "ncdf4", then just calling `print(nc)` will suffice. Objects of class "ncdf4" are returned from `nc_open` and `nc_create`.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

http://dwpierce.com/software

See Also

`ncvar_def`

Examples

```
## Not run:
# Open a netCDF file, print information about it
nc <- nc_open( "salinity.nc" )
print(nc)

## End(Not run)
```

Index