

Description

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

Details

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. NetCDF files contain one or more variables, which are structured as regular N-dimensional arrays. They also contain dimensions, which describe the extent of the variables' arrays. Data can be read from or written to variables in arbitrary hyperslabs. The R package 'ncdf' allows reading from, writing to, and creation of netCDF files. Note that the netCDF library must already be installed on your machine for this R interface to the library to work.

If you are absolutely 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 [open.ncdf](#) to open the file, then call [get.var.ncdf](#) to read the data from the file.

If you want to WRITE data to a new netCDF file, first call [dim.def.ncdf](#) to define the dimensions that your data exists along (for example, perhaps latitude and longitude), then call [var.def.ncdf](#) to define a variable in the netCDF file that will hold your data, then call [create.ncdf](#) to create the netCDF file, then call [put.var.ncdf](#) to write your data to the newly created netCDF file.

This is version 1.5 of the ncdf library.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[att.put.ncdf](#), [att.get.ncdf](#), [close.ncdf](#), [create.ncdf](#), [dim.def.ncdf](#), [get.var.ncdf](#), [put.var.ncdf](#), [open.ncdf](#), [print.ncdf](#), [set.missval.ncdf](#), [sync.ncdf](#), [var.def.ncdf](#), [redef.ncdf](#).

`put.var.ncdf`

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.

Usage

```
put.var.ncdf( nc, varid, vals, start=NA, count=NA, verbose=FALSE )
```

Arguments

<code>nc</code>	An object of class <code>ncdf</code> (as returned by either function <code>open.ncdf()</code> or function <code>create.ncdf()</code>), indicating what file to write to.
<code>varid</code>	What variable to write the data to. Can be a string with the name of the variable, an object of class <code>var.ncdf</code> , or the "id" field of a <code>var.ncdf</code> object.
<code>vals</code>	The values to be written.
<code>start</code>	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,...).
<code>count</code>	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 must be specified. As a special case, the value "-1" indicates that all entries along that dimension should be written.
<code>verbose</code>	If true, prints information while executing.

Details

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

Note that the type 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 `set.missval.ncdf` 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://www.unidata.ucar.edu/packages/netcdf/>

See Also

[dim.def.ncdf](#), [create.ncdf](#), [get.var.ncdf](#).

Examples

```
# Make a few dimensions we can use
nx <- 3
ny <- 4
nt <- 5
xvals <- (1:nx)*100.
dimX <- dim.def.ncdf( "X", "meters", xvals )
dimY <- dim.def.ncdf( "Y", "meters", (1:ny)*100. )
dimT <- dim.def.ncdf( "Time", "seconds", (1:nt)/100., unlim=TRUE )

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

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

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

data2d <- runif(nx*ny)
put.var.ncdf( nc, var2d, data2d )      # no start or count: write all values
# Write a 1-d slice to the 2d var
put.var.ncdf( 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
put.var.ncdf( 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)
  put.var.ncdf( nc, var3d, data2d, start=c(1,1,i), count=c(-1,-1,1) )

close.ncdf(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.
#-----
dimnchar  <- dim.def.ncdf("nchar",  "", 1:12 )
dimcolorno <- dim.def.ncdf("colorno", "", 1:nstrings )

varcolors <- var.def.ncdf("colors", "", list(dimnchar, dimcolorno),
                          NA, prec="char" )

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

put.var.ncdf( ncid, "colors", cnames )

close.ncdf( ncid )

```

`att.get.ncdf`

Get attribute from netCDF file

Description

Reads an attribute from a netCDF file.

Usage

```
att.get.ncdf( nc, varid, attname )
```

Arguments

<code>nc</code>	An object of class <code>ncdf</code> (as returned from <code>open.ncdf</code>), indicating what file to read from.
<code>varid</code>	The variable whose attribute is to be read. Can be a character string with the variable's name, an object of class <code>var.ncdf</code> , or an id contained in the "id" field of a <code>var.ncdf</code> object. As a special case, if <code>varid==0</code> , then it is assumed that we are reading a global attribute rather than a particular variable's attribute.
<code>attname</code>	Name of the attribute to read.

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

A list with two attributes, "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, then a double value is returned. If the on-disk type is character, then a character string is returned.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[att.put.ncdf](#).

Examples

```
# Make a simple netCDF file
filename <- "atttest_types.nc"
dim <- dim.def.ncdf( "X", "inches", 1:12 )
var <- var.def.ncdf( "Data", "unitless", dim, -1 )
ncnew <- create.ncdf( filename, var )

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

# Now illustrate the use of the att.get.ncdf function by reading them back in
doitfor <- function( nc, var, attname ) {
  av <- att.get.ncdf( 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 <- open.ncdf( filename )
var <- "Data"
doitfor( nc, var, "testatt_dbl" )
doitfor( nc, var, "testatt_single" )
doitfor( nc, var, "testatt_wacko" )
doitfor( nc, 0, "globalatt_int" )
doitfor( nc, 0, "globalatt_short" )
doitfor( nc, 0, "description" )

```

`att.put.ncdf` *Put an attribute into a netCDF file*

Description

Writes an attribute to a netCDF file.

Usage

```
att.put.ncdf( nc, varid, attname, attval, prec=NA, verbose=FALSE,
              definemode=FALSE )
```

Arguments

<code>nc</code>	An object of class <code>ncdf</code> (as returned from <code>open.ncdf</code>), indicating what file to write to.
<code>varid</code>	The variable whose attribute is to be written. Can be a character string with the variable's name, an object of class <code>var.ncdf</code> , or an id contained in the "id" field of a <code>var.ncdf</code> object. As a special case, if <code>varid==0</code> , then a global attribute is written instead of a variable's attribute.
<code>attname</code>	Name of the attribute to write.
<code>attval</code>	Attribute to write.
<code>prec</code>	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", "single", "double", or "text".
<code>verbose</code>	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.

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://www.unidata.ucar.edu/packages/netcdf/>

See Also

[att.put.ncdf](#), [att.text.get.ncdf](#).

Examples

```
# Make a simple netCDF file
filename <- "atttest_types.nc"
dim <- dim.def.ncdf( "X", "inches", 1:12 )
var <- var.def.ncdf( "Data", "unitless", dim, -1 )
ncnew <- create.ncdf( filename, var )

# Define some attributes of various types
attvaldbl <- 3.1415926536
att.put.ncdf( ncnew, var, "testatt_dbl", attvaldbl, prec="double" )
attvalsingle <- c(1.0,4.0,9.0,16.0)
att.put.ncdf( ncnew, var, "testatt_single", attvalsingle )
# varid=0 means it is a global attribute
att.put.ncdf( ncnew, 0, "globalatt_int", 32000, prec="int" )
att.put.ncdf( ncnew, 0, "globalatt_short", 7, prec="short" )
att.put.ncdf( ncnew, 0, "description",
              "this is a test file with attributes of various types")
close.ncdf(ncnew)
```

`close.ncdf`

Close a netCDF File

Description

Closes an open netCDF file, which flushes any unwritten data to disk.

Usage

```
close.ncdf( con, ... )
```

Arguments

`con` An object of class `ncdf` (as returned by either function `open.ncdf()` or function `create.ncdf()`), indicating what file to read from.

`...` Other arguments passed to or from other methods.

Details

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

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[sync.ncdf](#).

Examples

```
## Not run: nc <- open.ncdf("salinity.nc")
## Not run: data <- get.var.ncdf( nc )      # Read the "only" var in the file
## Not run: close.ncdf(nc)
```

Description

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

Usage

```
create.ncdf( filename, vars, verbose=FALSE )
```

Arguments

<code>filename</code>	Name of the netCDF file to be created.
<code>vars</code>	Either an object of class <code>var.ncdf</code> describing the variable to be created, or a vector of such objects to be created.
<code>verbose</code>	If TRUE, then information is printed while the file is being created.

Details

This routine creates a new netCDF file on disk. It must be given the variables in the file that will be created. Keep in mind that the new file may not actually be written to disk until `close.ncdf` is called.

Value

An object of class `ncdf`, which has the fields described in [open.ncdf](#).

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[dim.def.ncdf](#), [var.def.ncdf](#).

Examples

```
# Define an integer dimension
dimState <- dim.def.ncdf( "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 <- var.def.ncdf("Pop", "count", dimState, -1,
                      longname="Population", prec="integer")

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

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

close.ncdf(ncnew)
```

<code>dim.def.ncdf</code>	<i>Define a netCDF Dimension</i>
---------------------------	----------------------------------

Description

Defines a netCDF dimension. This dimension initially only exists in memory. It is added to a netCDF variable using `var.def.ncdf()`, and written to disk using `create.ncdf()`.

Usage

```
dim.def.ncdf( name, units, vals, unlim=FALSE )
```

Arguments

<code>name</code>	Name of the dimension to be created (character string).
<code>units</code>	The dimension's units (character string).
<code>vals</code>	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.
<code>unlim</code>	If TRUE, this dimension is unlimited.

Details

This routine creates a netCDF dimension in memory. The dimension can then be passed to the routine `var.def.ncdf()` when creating a variable.

Note that this interface to the netCDF library 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 requires 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. 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.

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 will be integer.

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

The `dim.ncdf` 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); 4) `len`, which is the length of this dimension; 5) `unlim`, which is a boolean indicating whether or not this is an unlimited dimension.

Value

An object of class `dim.ncdf` that can later be passed to `var.def.ncdf()`.

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 `'put.var.ncdf'`, 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://www.unidata.ucar.edu/packages/netcdf/>

See Also

[var.def.ncdf](#), [create.ncdf](#)

Examples

```
# Define some straightforward dimensions
x <- dim.def.ncdf( "Lon", "degreesE", 0.5:359.5)
y <- dim.def.ncdf( "Lat", "degreesN", as.double(-89:89))
t <- dim.def.ncdf( "Time", "days since 1900-01-01", 1:10, unlim=TRUE)

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

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

close.ncdf(ncnew)

# Now, illustrate some manipulations of the dim.ncdf object.
filename <- "salinity.nc"
nc <- open.ncdf( 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))
}
```

`enddef.ncdf`

Takes a netCDF file out of define mode

Description

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

Usage

```
enddef.ncdf( nc )
```

Arguments

`nc` An object of class `ncdf` (as returned by either function `open.ncdf()` or function `create.ncdf()`), indicating what file to read from.

Details

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.

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. In particular, the call to write an attribute ([att.put.ncdf](#)) handles this automatically. An example of the kind of situation where you would need this call is if you are adding a new variable to an already-existing netCDF file. This case is not really handled by this package.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[redef.ncdf](#).

Examples

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

<code>get.var.ncdf</code>	<i>Read data from a netCDF file</i>
---------------------------	-------------------------------------

Description

Reads data from an existing netCDF file.

Usage

```
get.var.ncdf(nc, varid=NA, start=NA, count=NA, verbose=FALSE,
signedbyte=TRUE, forcevarid=NA)
```

Arguments

<code>nc</code>	An object of class <code>ncdf</code> (as returned by either function <code>open.ncdf()</code> or function <code>create.ncdf()</code>), indicating what file to read from.
<code>varid</code>	What variable to read the data from. Can be a string with the name of the variable, an object of class <code>var.ncdf</code> , or the "id" field of a <code>var.ncdf</code> object. 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 <code>dim.ncdf</code> 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 <code>open.ncdf(filename, readunlim=FALSE)</code> , then read the dimension values in later with this function.
<code>start</code>	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,...).
<code>count</code>	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.
<code>verbose</code>	If TRUE, then progress information is printed.
<code>signedbyte</code>	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.
<code>forcevarid</code>	Internal use only. This indicates that the integer value passed in this argument is the actual variable ID to use, and no interpretation of the 'varid' argument is done.

Details

This routine reads data values from a variable in a netCDF file. The file must already have been opened with `open.ncdf`.

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 `set.missval.ncdf` 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".

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[put.var.ncdf](#).

Examples

```
# Start with the simplest example.  If the file only has one variable in it,
# you can read the data as easily as this:
#
nc <- open.ncdf("salinity.nc")
# NOTE how not specifying varid reads the "only" var in the file
data <- get.var.ncdf( nc )
close.ncdf(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 "put.var.ncdf".  We open the
# file with readunlim=FALSE for potentially faster access, and to illustrate
# (below) how to read in the unlimited dimension values.
#
nc <- open.ncdf( "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$var[[1]]
data1 <- get.var.ncdf( nc, v1 ) # by default, reads ALL the data
print(paste("Data for var ",v1$name,":",sep=""))
print(data1)

# This shows how the shape of the read data is preserved
v2 <- nc$var[[2]]
data2 <- get.var.ncdf( nc, v2 )
print(paste("Var 2 has name",v2$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 timestep
  data3 <- get.var.ncdf( nc, v3, start=start, count=count )

  # Now read in the value of the timelike dimension
  timeval <- get.var.ncdf( 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)
}

close.ncdf(nc)

```

ncdf-internal

Internal ncdf functions

Description

Internal ncdf functions.

Details

These are not to be called by the user.

open.ncdf

Open a netCDF File

Description

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

Usage

```
open.ncdf( con, write=FALSE, readunlim=TRUE, verbose=FALSE, ... )
```

Arguments

<code>con</code>	Name of the existing netCDF file to be opened.
<code>write</code>	If FALSE (default), then the file is opened read-only. If TRUE, then writing to the file is allowed.
<code>readunlim</code>	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 <code>get.var.ncdf()</code>).
<code>...</code>	Arguments passed to or from other methods.
<code>verbose</code>	If TRUE, then messages are printed out during execution of this function.

Details

This routine opens an existing netCDF file for reading (or, if `write=TRUE`, for writing). To create a new netCDF file, use `create.ncdf()` 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 `ncdf`. 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 `dim.ncdf`; 7) `var`, which is a list of objects of class `var.ncdf`; 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 `ncdf` object returned by this function, as well as the list of `dim.ncdf` objects contained in the `ncdf` object's "dim" list and the `var.ncdf` 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 `ncdf` object or its children to get information about the netCDF file. (Note: the `dim.ncdf` object is described in the help file for `dim.def.ncdf`; the `var.ncdf` object is described in the help file for `var.def.ncdf`).

Value

An object of class `ncdf` that has the fields described below.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[dim.def.ncdf](#), [var.def.ncdf](#).

Examples

```
# Define an integer dimension
dimState <- dim.def.ncdf( "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 <- var.def.ncdf("Pop", "count", dimState, -1,
                      longname="Population", prec="integer")

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

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

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

close.ncdf(ncnew)

# Now open the file and read its data
ncold <- open.ncdf("states_population.nc")
data <- get.var.ncdf(ncold)
print("here is the data in the file:")
print(data)
close.ncdf( ncold )
```

`print.ncdf`

Print Information About a netCDF File

Description

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

Usage

```
print.ncdf( x, ... )
```

Arguments

`x` An object of class "ncdf".
`...` 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 "ncdf", then just calling `print(nc)` will suffice. Objects of class "ncdf" are returned from `open.ncdf`.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[var.def.ncdf](#)

Examples

```
# Open a netCDF file, print information about it
nc <- open.ncdf( "salinity.nc" )
print(nc)
```

<code>redef.ncdf</code>	<i>Puts a netCDF file back into define mode</i>
-------------------------	---

Description

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

Usage

```
redef.ncdf( nc )
```

Arguments

`nc` An object of class `ncdf` (as returned by either function `open.ncdf()` or function `create.ncdf()`), indicating what file to read from.

Details

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.

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. In particular, the call to write an attribute ([att.put.ncdf](#)) handles this automatically. An example of the kind of situation where you would need this call is if you are adding a new variable to an already-existing netCDF file. This case is not really handled by this package.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

[enddef.ncdf](#).

Examples

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

<code>set.missval.ncdf</code>	<i>Set the Missing Value Attribute For a netCDF Variable</i>
-------------------------------	--

Description

Sets the `missing_value` attribute for a netCDF variable.

Usage

```
set.missval.ncdf( nc, varid, missval )
```

Arguments

<code>nc</code>	An object of class <code>ncdf</code> , as returned by <code>open.ncdf</code> or <code>create.ncdf</code> .
<code>varid</code>	The name, <code>var.ncdf</code> object, or variable ID whose missing value will be set.
<code>missval</code>	The missing value to set.

Details

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 `var.def.ncdf`. 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://www.unidata.ucar.edu/packages/netcdf/>

See Also

[var.def.ncdf](#).

Examples

```
# Make an example netCDF file with a given missing value. We will
# then change the missing value in the file using set.missval.ncdf.

origMissVal <- -1.
dimX <- dim.def.ncdf( "X", "meters", 1:7 )
varAlt <- var.def.ncdf( "Altitude", "km", dimX, origMissVal )
ncnew <- create.ncdf( "transect.nc", varAlt )
data <- c(10.,2.,NA,1.,7.,NA,8.)
put.var.ncdf( ncnew, varAlt, data )
close.ncdf(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 <- open.ncdf( "transect.nc", write=TRUE )
varname <- "Altitude"
data <- get.var.ncdf( nc, varname ) # data now has: 10., 2., NA, 1., 7., NA, 8.
print(data)
set.missval.ncdf( nc, varname, newMissVal )
put.var.ncdf( nc, varname, data )
close.ncdf(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 files
# data and change the values to the new missing value.

```

sync.ncdf

Synchronize (flush to disk) a netCDF File

Description

Flushes any pending operations on a netCDF file to disk.

Usage

```
sync.ncdf( nc )
```

Arguments

nc An object of class `ncdf` (as returned by either function `open.ncdf()` or function `create.ncdf()`), indicating what file to read from.

Details

Data in a netCDF file might be 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://www.unidata.ucar.edu/packages/netcdf/>

Examples

```
# 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 <- dim.def.ncdf( "X", "meters", 1:nx )
dimy <- dim.def.ncdf( "Y", "meters", 1:ny )
dimt <- dim.def.ncdf( "Time", "days since 1900-01-01", 0, unlim=TRUE )

vartemp <- var.def.ncdf( "Temperature", "degC", list(dimx,dimy,dimt), 1.e30 )
nc <- create.ncdf( "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
  put.var.ncdf( 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
  put.var.ncdf( nc, dimt, timeval, start=i, count=1 )

  # Flush this timesteps data to the file so we dont lose it
  # if there is a crash or other problem
  sync.ncdf( nc )
}

# Always remember to close the file when done!!
close.ncdf(nc)
```

Description

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

Usage

```
var.def.ncdf( name, units, dim, missval, longname=name, prec="single")
```

Arguments

<code>name</code>	Name of the variable to be created (character string).
<code>units</code>	The variable's units (character string).
<code>dim</code>	The variable's dimension(s) (one or a list of "dim.netcdf" class objects).
<code>missval</code>	The variable's missing value.
<code>longname</code>	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"
<code>prec</code>	Precision of the created variable. Valid options: 'short' 'integer' 'single' 'double' 'char'.

Details

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

Note that this interface to the netCDF library includes that 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 `create.ncdf`, then data values for the variable can be written to disk using `put.var.ncdf`.

This function returns a `var.ncdf` object, which describes the newly-created variable. However, the `var.ncdf` object is used for more than just creating new variables. The function `open.ncdf` returns a `ncdf` object that itself contains a list of `var.ncdf` 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 `dim.ncdf` object instead.)

The `var.ncdf` 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 "dim.ncdf" (see `dim.def.ncdf`), 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.

Value

An object of class `var.ncdf` that can later be passed to `create.ncdf()`.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>

See Also

`dim.def.ncdf`, `create.ncdf`, `put.var.ncdf`.

Examples

```
# Define an integer dimension
dimState <- dim.def.ncdf( "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 <- var.def.ncdf("Pop", "count", dimState, -1,
                      longname="Population", prec="integer")

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

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

```

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

close.ncdf(ncnew)

# Now illustrate some manipulations of the var.ncdf object
filename <- "states_population.nc"
nc <- open.ncdf(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 <- dim.def.ncdf( "Time", "days", 1:3 )
missval <- -1
varShort <- var.def.ncdf( "varShort", "meters", dimt, missval, prec="short")
varInt <- var.def.ncdf( "varInt", "meters", dimt, missval, prec="integer")
varFloat <- var.def.ncdf( "varFloat", "meters", dimt, missval, prec="single")
varDouble<- var.def.ncdf( "varDouble","meters", dimt, missval, prec="double")
nctypes <- create.ncdf("vartypes.nc", list(varShort,varInt,varFloat,varDouble) )
close.ncdf(nctypes)

```

```
version.ncdf
```

Report version of ncdf library

Description

Returns a string that is the version number of the ncdf library

Usage

```
version.ncdf()
```

Arguments

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 netcdf library.

Author(s)

David W. Pierce <dpierce@ucsd.edu>

References

<http://www.unidata.ucar.edu/packages/netcdf/>