RNetCDF – A Package for Reading and Writing NetCDF Datasets

by Pavel Michna and Milton Woods

Abstract This paper describes the **RNetCDF** package (version 1.6), an interface for reading and writing files in Unidata NetCDF format, and gives an introduction to the NetCDF file format. NetCDF is a machine independent binary file format which allows storage of different types of array based data, along with short metadata descriptions. The package presented here allows access to the most important functions of the NetCDF C-interface for reading, writing, and modifying NetCDF datasets. In this paper, we present a short overview on the NetCDF file format and show usage examples of the package.

Introduction

NetCDF is a widely used file format in atmospheric and oceanic research – especially for weather and climate model output – which allows storage of different types of array based data, along with a short data description. The NetCDF format (Network Common Data Format, `http://www.unidata.ucar.edu/software/netcdf/`) has been developed since 1988 by Unidata (a programme sponsored by the United States National Science Foundation) with the main goal of making best use of atmospheric and related data for education and research (Rew et al., 2011; Rew and Davis, 1990).

NetCDF files are stored as machine-independent binary data, such that files can be exchanged between computers without explicit conversion (Rew and Davis, 1990). Until version 3.6.0, only one binary data format was used. This is the default format for all NetCDF versions and is also named NetCDF classic format (Rew et al., 2011). Version 3.6.0 of the NetCDF library introduced the 64-bit offset format, which allowed addressing of much larger datasets; version 4.0.0 introduced also the HDF5 format, in a way that the NetCDF library can use HDF5 as its external format. By default, however, the classic format is still used (Rew et al., 2011).

One particular advantage of NetCDF over some other binary formats, such as the RData format used by R, is the ability to access and modify arbitrary sections of array data. This allows massive datasets to be processed efficiently, even if they are larger than the virtual memory available on a particular system. To reduce disk space requirements, floating-point values are often packed into 8- or 16-bit integers, and the NetCDF-4 (HDF5) format supports transparent compression using the zlib library.

RNetCDF (Michna, 2012) was designed to handle the classic format and is also able to read and write files with 64-bit offset. If **RNetCDF** is compiled and linked with version 4.0.0 or later of the NetCDF library, files in NetCDF-4 (HDF5) binary format can be read if they use the data model of NetCDF-3 or earlier. In this paper we give a short overview of the concept of NetCDF based on Unidata's reference manuals, followed by the concept of the package and usage examples.

What are NetCDF datasets?

Data model

A NetCDF dataset contains dimensions, variables, and attributes, each identified both by a name and an ID number. These components can be used together to capture the meaning of data and relations among data fields in an array-oriented dataset. The NetCDF library allows simultaneous access to multiple NetCDF datasets which are identified by dataset ID numbers, in addition to ordinary file names.

A NetCDF dataset contains a symbol table for variables containing their name, data type, rank (number of dimensions), dimensions, and starting disk address. Each element is stored at a disk address which is a linear function of the array indices (subscripts) by which it is identified. Hence, these indices need not be stored separately (as in a relational database). This provides a fast and compact storage method (Rew et al., 2006). The advantage of the NetCDF library is that there is no need for the user to take care of the physical representation of multidimensional data on the disk.

Name	Length	Description	Limits		
`char`	8-bit	characters intended for representing text			
`byte`	8-bit	signed or unsigned integers	-128	...	$+127$
`short`	16-bit	signed integers	$-32'768$	...	$+32'767$
`int`	32-bit	signed integers	$-2'147'483'648$	...	$+2'147'483'647$
`float`	32-bit	IEEE floating-point (6 significant digits)	$\pm 1.175 \times 10^{-38}$	...	$\pm 3.403 \times 10^{38}$
`double`	64-bit	IEEE floating-point (15 significant digits)	$\pm 2.225 \times 10^{-308}$	...	$\pm 1.798 \times 10^{308}$

Table 1: External data types which are supported by the NetCDF interface.

Data types

The NetCDF interface defines six primitive external data types – `char`, `byte`, `short`, `integer`, `float`, and `double` (Rew et al., 2006). Their exact representation is shown in Table 1. These types were chosen to provide a reasonably wide range of trade-offs between data precision and number of bits required for each value. These external data types are independent of whatever internal data types are supported by a particular machine and language combination. The basic unit of named data in a NetCDF dataset is a variable (Rew et al., 2006). When a variable is defined, its shape is specified as a list of dimensions. These dimensions must already exist at the time of definition of a variable.

Dimensions

A dimension may be used to represent a real physical dimension, for example, time, latitude, longitude, or height. A dimension might also be used to index other quantities, for example station or model-run-number (Rew et al., 2006).

A NetCDF dimension has both a name and a length, where the dimension length is an arbitrary positive integer starting at 1. One dimension in a NetCDF dataset can be of unlimited length. Such a dimension is called the unlimited dimension or the record dimension. A variable with an unlimited dimension can grow to any length along that dimension. The unlimited dimension index is like a record number in conventional record-oriented files. A NetCDF dataset can have at most one unlimited dimension, but need not have any. If a variable has an unlimited dimension, that dimension must be the most significant (slowest changing) one.

Variables

Variables are used to store the bulk of the data in a NetCDF dataset. A variable represents an array of values of the same type. A scalar value is treated as a 0-dimensional array. A variable has a name, a data type, and a shape described by its list of dimensions specified when the variable is created. A variable may also have associated attributes, which may be added, deleted or changed after the variable is created (Rew et al., 2006). The shape of a variable cannot be changed after definition, only growing along the unlimited dimension is possible. Missing values (NA) have no internal representation. For this purpose, a respective attribute has to be defined for each variable. Most applications (including **RNetCDF**) accept the names '`_FillValue`' and '`missing_value`', although the latter is deprecated and should not be used when creating new datasets.

Handling of strings

NetCDF does not have a primitive string type, but does have arrays of type `char`, each of which is 8 bits in size. The main difference is that strings are arrays of chars of variable length, while char arrays are of fixed length (Rew et al., 2006). If an array of strings has to be created (e.g., a list of station names), internal routines read char arrays and convert them to strings without requiring the user to deal with trailing zeroes or padding. The zero-byte termination of strings is done automatically, and the user only needs to ensure that the fastest varying dimension (often named '`max_string_length`') is long enough to contain the terminating zero-byte, i.e., `max(nchar(my_strings))+1` where `my_strings` is the data to be written.

Attributes

NetCDF attributes are used to store metadata, similar in many ways to the information stored in data dictionaries and schema in conventional database systems. Most attributes provide information about

a specific variable. These are identified by the name (or ID) of that variable, together with the name of the attribute. An attribute has an associated variable (the null variable for global attributes), a name, a data type, a length, and a value (Rew et al., 2006). Most generic applications that process NetCDF datasets assume standard attribute conventions (see below) and it is strongly recommended that these be followed unless there are good reasons for not doing so.

Naming conventions

There are almost no restrictions on how a NetCDF dataset should be named and structured. However, there are different conventions like COARDS and CF (`http://cf-pcmdi.llnl.gov/`, Eaton et al. 2011), and it is highly recommended to follow at least the basic practice to ensure portability and self-description of the contents. Variable, dimension and attribute names should begin with a letter and be composed of letters (case significant), digits, and underscores. The CF-convention (NetCDF Climate and Forecast Metadata Convention) permits neither the use of the hyphen character, nor leading underscores in names. Finally, NetCDF files should have the file name extension '`.nc`'.

Coordinate systems

A *coordinate variable* is a one-dimensional variable with the same name as a dimension, which names the coordinate values of the dimension. It should not contain any missing data (for example, no '`_FillValue`' or '`missing_value`' attributes) and must be strictly monotonic (values increasing or decreasing). A variable's *coordinate system* is the set of coordinate variables used by the variable. It is good practice to respect the following rules (Rew et al., 2006):

- Create coordinate variables for every dimension (except for string length dimensions).
- Give each coordinate variable at least '`unit`' and '`long_name`' attributes to document its meaning.
- Share dimensions to indicate that two variables use the same coordinates along that dimension. If two variables' dimensions are not related, create separate dimensions for them, even if they happen to have the same length.

In climatological applications, often geographical coordinates are used. Variables representing latitude must always explicitly include the '`units`' attribute; there is no default value. The recommended unit of latitude is '`degrees_north`', and '`degrees_east`' for longitude (Eaton et al., 2011).

Handling of time

There is no single way to deal with time in NetCDF datasets, but in most cases, time definitions from Unidata's UDUNITS library are used (see `http://www.unidata.ucar.edu/software/udunits`). Variables representing time must always explicitly include the '`units`' attribute; there is no default value. The '`units`' attribute takes a string value formatted as per the recommendations in the UDUNITS package (Eaton et al., 2011), usually in the form '`time_units since time_reference`'.

The most commonly used time units (and their abbreviations) include '`day`', '`hour`', '`minute`' and '`second`' or their plural forms. The units '`year`' and '`month`' may also be used, but they refer to fractional numbers of days related to successive passages of the sun through the vernal equinox. It may be preferable to use units related to the calendar year, including a '`common_year`' of 365 days, a '`leap_year`' of 366 days, a '`Julian_year`' of 365.25 days, or a '`Gregorian_year`' of 365.2425 days.

A reference time string is required to appear after the identifier '`since`', and it may include date alone, date and time, or date, time and time zone. An example of a valid reference time is '`1970-1-1 00:00:00 10:00`', which is midnight on January 1st, 1970 in a time zone that is 10 hours east of Coordinated Universal Time (such as Australian Eastern Standard Time).

Implementation

RNetCDF enables most of the functionality of the NetCDF C-interface (version 3.6.1) to be called from within R. Because time is often stored as a numeric vector with a reference time and unit according to Unidata's UDUNITS library, calendar conversion functions from UDUNITS are also included in the package.

The programming interfaces provided by **RNetCDF** will be familiar to developers who have used NetCDF from compiled languages such as Fortran and C. An alternative package, **ncdf** (Pierce, 2011) and its successor **ncdf4** (Pierce, 2013), provides a higher-level interface to NetCDF that may be preferred by some users, but it does not allow deleting and renaming of attributes. However, the

lower-level functions in **RNetCDF** allow users to define functions and data structures that match their purposes. Although a high-level interface generally requires less work by users, we believe that **RNetCDF** provides more and better functionality, since users need not care about technical issues at the C level, yet they still have the means to perform nearly all operations that are possible on NetCDF datasets. We have included one higher-level function that is not part of the C-interface which enables reading of a whole NetCDF dataset using one command, which is a common task when working with such datasets.

All six of the external data types shown in Table 1 are supported. However, when reading data into R, only the R data types `character` and `numeric` will be distinguished. The NetCDF C-library converts integer and floating point values in a NetCDF file to double precision values in R, and the reverse conversions are performed during write operations. Reading and writing of data arrays is done by specifying a corner and a vector of edge lengths. The capabilities of the package are restricted to consecutive read/write; subsampling and mapping are not currently supported by **RNetCDF** but they can be performed easily using standard R commands.

The classic and 64-bit NetCDF file formats store metadata, such as dimensions, variable names and attributes, in a binary header at the start of the file. The contents of variables are stored after the header, and little or no padding is used to separate the sections of the file. If the metadata of the file are changed after variables are written, it is likely that variables will need to be moved within the file to accommodate a change in the size of the header. To avoid the overhead of such data movement, the usual approach is to define all of the metadata before writing data to variables. The NetCDF C-library uses distinct modes for defining metadata and writing variables, and special routines are used to switch between these two modes. However, for the sake of simplicity, the mode-switching routines are hidden by the **RNetCDF** interface, and the appropriate mode is selected for each operation requested.

Usage examples

Creating a NetCDF dataset

As an example, assume we have a climatological dataset with daily temperature measurements at five imaginary weather stations. Three variables are defined: time (as date with year, month, day, hour, minute, second), temperature and station name:

```
mytime        <- matrix(nrow=2, ncol=6)
mytime[1,]    <- c(2012, 06, 01, 12, 00, 00)
mytime[2,]    <- c(2012, 06, 02, 12, 00, 00)
mytime_units  <- "days since 1970-01-01 00:00:00"

mytemperature <- matrix(c(1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, NA, NA, 9.9),
                        ncol=2, nrow=5)
myname        <- c("Alfa", "Bravo", "Charlie", "Delta", "Echo")
```

When creating the NetCDF dataset, the organisation of the data should be known in advance. While changes to the structure of the file are possible, they may involve significant reorganisation of data within the file. To allow for expansion of a file with new data, it is possible to declare a single dimension with "unlimited" size. As a first step in our example, the file has to be created and all dimensions and variables need to be defined:

```
nc <- create.nc("foo.nc")

dim.def.nc(nc, "station", 5)
dim.def.nc(nc, "time", unlim=TRUE)
dim.def.nc(nc, "max_string_length", 32)

var.def.nc(nc, "time", "NC_INT", "time")
var.def.nc(nc, "temperature", "NC_DOUBLE", c("station", "time"))
var.def.nc(nc, "name", "NC_CHAR", c("max_string_length", "station"))
```

At this point, missing values (NA) cannot be written and the time axis is not yet defined. For this purpose, attributes have to be set and the time matrix needs to be converted into a vector with a reference time (as defined already above):

```
att.put.nc(nc, "temperature", "_FillValue", "NC_DOUBLE", -99999.9)
att.put.nc(nc, "time", "units", "NC_CHAR", mytime_units)
```

```
mytime_ut <- utinvcal.nc(mytime_units, mytime)
```

Now the variable data can be written. To ensure that the data are written to the file and not buffered in memory, the file should be closed when all operations are complete:

```
var.put.nc(nc, "name", myname)
var.put.nc(nc, "time", mytime_ut)
var.put.nc(nc, "temperature", mytemperature)
close.nc(nc)
```

If more data is to be added to the file in the same R session, the file may be left open, but to avoid loss of data, it may be desirable to force the flushing of buffers to disk using the function `sync.nc()` at critical stages of a calculation.

In our example, the NetCDF dataset is written to disk with the absolute minimum of required attributes. However, such a dataset is not really self-describing and would not conform with any conventions. Therefore, further attributes would need to be set. According to the CF-standard, a variable should have at least the attributes '`long_name`' (e.g., '`measured air temperature`'), '`units`' (e.g., '`degrees_celsius`'), and '`standard_name`' (e.g., '`air_temperature`') (the latter is not needed for the time coordinate variable). The possible values for '`standard_name`' can be found in the CF conventions document. CF also requests the indication of six global attributes, namely '`title`', '`history`', '`institution`', '`source`', '`comment`', and '`references`'. Although not mandatory, it is recommended that NetCDF datasets comply with the CF or any other standard, so that the contents of a file are described unambiguously. If these rules are followed, NetCDF datasets can be explored and processed using general-purpose software, and they can be distributed or archived without any risk that the data in a file could become separated from its description.

Reading an existing NetCDF dataset

To show the contents of a NetCDF dataset, it must first be opened with the `open.nc()` function. The `print.nc()` function displays an overview of the dataset on standard output, giving the dimension definitions, variable definitions including their attributes, and the contents of the global attributes. For the example dataset created earlier, an overview can be displayed as follows:

```
nc <- open.nc("foo.nc")
print.nc(nc)
```

The contents of a single variable can be read from a NetCDF dataset using the `var.get.nc()` function. For a variable that contains a large array of data, it may be desirable to read only an array section from the variable, which can be accomplished by specifying a start index and number of elements for each dimension of the array, as demonstrated below. Notice that the optional `start` and `count` arguments are vectors with one element for each dimension. Where the `count` argument has a value of NA, the corresponding dimension is read in full.

```
mytemp <- var.get.nc(nc, "temperature", start=c(NA,2), count=c(NA,1))
```

The easiest way to read the contents of all variables from a NetCDF dataset is by using the function `read.nc()`, which is available in **RNetCDF** version 1.6 or later. This function returns a list with the variables as named elements. Although this function has no equivalent in the NetCDF C-interface, it has been added to **RNetCDF** to simplify a common operation. For example, the contents of all variables can be read from our example dataset and the '`temperature`' variable copied to another variable using the following commands:

```
nc_data <- read.nc(nc)
mytemp  <- nc_data$temperature
```

Attributes can be read from variables that are identified by name or number, and global attributes can be read using the special variable name '`NC_GLOBAL`'. For example, conversion of relative times into calendar times requires the '`units`' attribute from the '`time`' variable, which may be read using the `att.get.nc()` function:

```
time_units <- att.get.nc(nc, "time", "units")
```

The NetCDF C-library provides a comprehensive set of functions to determine the structure of a NetCDF dataset, including the names and sizes of dimensions and variables. These functions can be used to write programs that handle NetCDF datasets without prior knowledge of their contents. Most

of the inquiry functions of the C-library are accessible through the **RNetCDF** functions `file.inq.nc()`, `dim.inq.nc()`, `var.inq.nc()` and `att.inq.nc()`, which provide detailed information about datasets, dimensions, variables and attributes, respectively. For example, the names of all dimensions in a NetCDF dataset can be determined as shown below. Note that NetCDF dimensions can be referenced by integers that are sequential from 0; the same applies to variables and attributes.

```
ndims    <- file.inq.nc(nc)$ndims
dimnames <- character(ndims)
for(i in seq_len(ndims)) {
    dimnames[i] <- dim.inq.nc(nc, i-1)$name
}
```

Packed variables

To reduce the space required for storage of NetCDF datasets, the CF-convention allows variables to be stored in a packed format. The values are stored in a variable with lower precision than the original data. For example, 32-bit floating point values are often converted to 16-bit integers, so that the file size is approximately halved. To minimise the loss of information caused by the conversion, the original values are shifted and scaled so that they span the range of the new data type.

The packing algorithm can be expressed as follows:

$$x_s = (\max x - \min x)/(\max y - \min y) \quad (1)$$
$$x_o = \min x - x_s \min y \quad (2)$$
$$y = (x - x_o)/x_s, \quad (3)$$

where x is the original data and y is the packed variable. The values of x_o and x_s are stored with the packed variable in the standard attributes '`add_offset`' and '`scale_factor`', respectively. These attributes allow the packing operation to be reversed, although the unpacked data will usually have less precision than the original values.

Versions 1.6 or later of **RNetCDF** provide options to convert packed variables during reading and writing. Functions `var.get.nc()` and `var.put.nc()` have optional arguments `unpack` and `pack` respectively, although they have default values of `FALSE` to ensure compatibility with previous versions. The newly released function **read.nc** also has an optional `unpack` argument, which has the default value of `TRUE` to provide easy access to most datasets. It should be noted that the `pack` and `unpack` options are only honoured for variables that define both of the attributes '`add_offset`' and '`scale_factor`'.

In the example considered previously, the temperature data could be stored in a packed variable during creation of the dataset as follows:

```
var.def.nc(nc,"temp_p","NC_SHORT", c("station", "time"))
att.put.nc(nc, "temp_p", "_FillValue", "NC_SHORT", -32767)

tmax   <- max(mytemperature, na.rm=TRUE)
tmin   <- min(mytemperature, na.rm=TRUE)
ymax   <-  32766
ymin   <- -32766
scale  <- (tmax-tmin)/(ymax-ymin)
offset <- tmin-ymin*scale

att.put.nc(nc, "temp_p", "add_offset", "NC_DOUBLE", offset)
att.put.nc(nc, "temp_p", "scale_factor", "NC_DOUBLE", scale)

var.put.nc(nc, "temp_p", mytemperature, pack=TRUE)
```

Calendar functions

The two calendar functions `utcal.nc()` and `utinvcal.nc()` of the package (converting time from arbitrary units into a UTC-referenced date and time, and vice versa) have the option to read/write date and time directly in string form. When reading such strings, the structure must be exactly '`YYYY-MM-DD hh:mm:ss`'.

```
> utcal.nc("days since 2012-01-01 00:00:00", c(0,1))

     year month day hour minute second
```

```
[1,] 2012     1   1    0      0      0
[2,] 2012     1   2    0      0      0
```

It is also possible to specify another timezone as the reference time, as shown in the following example using Central European Time (CET):

```
> utcal.nc("days since 2012-01-01 00:00 +01:00", c(0,1))

     year month day hour minute second
[1,] 2011    12  31   23      0      0
[2,] 2012     1   1   23      0      0
```

If a user needs to have the date and time information as a string, the `type` argument can be set appropriately:

```
> utcal.nc("days since 2012-01-01 00:00 +01:00", c(0,1), type="s")

[1] "2011-12-31 23:00:00" "2012-01-01 00:00:00"
```

This functionality is intended especially for extracting axis descriptions in an efficient manner. Formatting of the string is possible using R functions for strings. For example, `substr()` can be used to extract the date or time components of the time-stamp.

Summary and outlook

RNetCDF is an R interface to the NetCDF C-library. Most of the functions provided by version 3 of NetCDF are accessible through **RNetCDF** in a way that allows users to build functions easily for their specific needs. Some higher-level features for frequently used operations are provided by **RNetCDF**, such as automatic support for missing values and packed variables and the ability to read all variables into an R list. Calendar conversion functions from Unidata's UDUNITS library are also included in this package to simplify the handling of time variables in NetCDF datasets.

Further information can be obtained in the **RNetCDF** reference manual and help pages (available from CRAN), Unidata's documentation for NetCDF (http://www.unidata.ucar.edu/software/netcdf/docs/) and UDUNITS (http://www.unidata.ucar.edu/software/udunits/), and the CF conventions documentation site (http://cf-pcmdi.llnl.gov/documents/).

The plans for future development include an option to read and write POSIXt time variables, which are used by many R routines, with automatic translation to and from the time format used in NetCDF datasets. The next major update will include support for the extended data model of NetCDF-4. However, a first analysis of the full NetCDF-4/HDF5 data model revealed that it might be difficult to map user defined data types (e.g., 6-bit structures) in a straight-forward way in R, so an intensive analysis of the new data model and the requirements of R users will be needed.

Readers who are interested in contributing to the development of **RNetCDF** are invited to contact the authors.

Acknowledgements

Brian Ripley and Uwe Ligges are gratefully acknowledged for making the Windows version possible, Simon Urbanek for enabling a binary distribution for Mac OS X.

Bibliography

B. Eaton, J. Gregory, B. Drach, K. Taylor, and S. Hankin. *NetCDF Climate and Forecast (CF) Metadata Conventions, Version 1.6*, 2011. [p3]

P. Michna. *RNetCDF: R Interface to NetCDF Datasets*, 2012. URL http://CRAN.R-project.org/package=RNetCDF. R Package Version 1.6.1-2. [p1]

D. Pierce. *ncdf: Interface to Unidata netCDF data files*, 2011. URL http://CRAN.R-project.org/package=ncdf. R Package Version 1.6.6. [p3]

D. Pierce. *ncdf4: Interface to Unidata netCDF (version 4 or earlier) format data files*, 2013. URL http://CRAN.R-project.org/package=ncdf4. R Package Version 1.10. [p3]

R. Rew and G. Davis. NetCDF: An interface for scientific data access. *IEEE Computer Graphics and Applications*, 10(4):76–82, 1990. [p1]

R. Rew, G. Davis, S. Emmerson, H. Davies, and E. Hartnett. *The NetCDF Users Guide, Version 3.6.1*. Unidata Program Center, 2006. [p1, 2, 3]

R. Rew, G. Davis, S. Emmerson, H. Davies, E. Hartnett, and D. Heimbigner. *The NetCDF Users Guide, Version 4.1.3*. Unidata Program Center, 2011. [p1]

Pavel Michna
Institute of Geography
Hallerstrasse 12, CH-3012 Bern
Switzerland
michna@giub.unibe.ch

Milton Woods
Bureau of Meteorology
GPO Box 1289, Melbourne VIC 3001
Australia
M.Woods@bom.gov.au