Package 'ncdfCF'

May 29, 2025

Type Package

Title Easy Access to NetCDF Files with CF Metadata Conventions

Version 0.6.0

Description Network Common Data Form ('netCDF') files are widely used for scientific data. Library-level access in R is provided through packages 'RNetCDF' and 'ncdf4'. Package 'ncdfCF' is built on top of 'RNetCDF' and makes the data and its attributes available as a set of R6 classes that are informed by the Climate and Forecasting Metadata Conventions. Access to the data uses standard R subsetting operators and common function forms.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports abind, CFtime (>= 1.6), methods, R6, RNetCDF, stringr

Collate 'AOI.R' 'AOImethod.R' 'CFArray.R' 'CFAuxiliaryLongLat.R' 'CFAxis.R' 'CFAxisCharacter.R' 'CFAxisDiscrete.R' 'CFAxisLatitude.R' 'CFAxisLongitude.R' 'CFAxisNumeric.R' 'CFAxisTime.R' 'CFAxisVertical.R' 'CFBounds.R' 'CFCellMeasure.R' 'CFDataset.R' 'CFGridMapping.R' 'CFLabel.R' 'NCObject.R' 'CFObject.R' 'CFResource.R' 'CFVariable.R' 'CFVariableBase.R' 'CFVariableL3b.R' 'NCDimension.R' 'NCGroup.R' 'NCUDT.R' 'NCVariable.R' 'makeCFObjects.R' 'ncdfCF-package.R' 'readCF.R' 'utils.R' 'wkt2.R' 'zzz.R'

Suggests data.table, knitr, rmarkdown, terra

VignetteBuilder knitr

Depends R (>= 3.5)

URL https://github.com/pvanlaake/ncdfCF

BugReports https://github.com/pvanlaake/ncdfCF/issues

NeedsCompilation no

Author Patrick Van Laake [aut, cre, cph]

Maintainer Patrick Van Laake <patrick@vanlaake.net>

Repository CRAN

Date/Publication 2025-05-29 09:40:02 UTC

Contents

aoi *Area of Interest*

Description

This function constructs the area of interest of an analysis. It consists of an extent and a resolution of longitude and latitude, all in decimal degrees.

The AOI is used to define the subset of data to be extracted from a data variable that has an auxiliary longitude-latitude grid (see the CFAuxiliaryLongLat class) at a specified resolution. The data variable thus has a primary coordinate system where the horizontal components are not a geographic system of longitude and latitude coordinates.

Usage

```
aoi(lonMin, lonMax, latMin, latMax, resX, resY)
```

Arguments

`lonMin`, `lonMax`, `latMin`, `latMax`
: The minimum and maximum values of the longitude and latitude of the AOI, in decimal degrees. The longitude values must agree with the range of the longitude in the data variable to which this AOI will be applied, e.g. `[-180,180]` or `[0,360]`.

`resX`, `resY`
: The separation between adjacent grid cell, in the longitude and latitude directions respectively, in decimal degrees. The permitted values lie within the range `[0.01 ... 10]`. If `resY` is missing it will use the value of `resX`, yielding square grid cells.

Details

Following the CF Metadata Conventions, axis coordinates represent the center of grid cells. So when specifying `aoi(20, 30, -10, 10, 1, 2)`, the south-west grid cell coordinate is at `(20.5, -9)`. If the axes of the longitude-latitude grid have bounds, then the bounds will coincide with the AOI and the `CFVariable$subset()` method that uses the AOI will attach those bounds as attributes to the resulting array.

If no resolution is specified, it will be determined from the separation between adjacent grid cells in both longitude and latitude directions in the middle of the area of interest. If no extent is specified (meaning, none of the values; if some but not all values are specified an error will be thrown), then the whole extent of the variable is used, extended outwards by the bounds if they are set or half the resolution otherwise. Thus, to get the entire extent of the variable but in a longitude-latitude grid and with a resolution comparable to the resolution at the original Cartesian coordinate system of the variable, simply pass `aoi()` as an argument to CFVariable$subset(). Note that any missing arguments are calculated internally and stored in the returned object, but only after the call to `CFVariable$subset()`.

Caching:

In data collections that are composed of multiple data variables in a single netCDF resource, a single auxiliary longitude-latitude grid may be referenced by multiple data variables, such as in ROMS data which may have dozens of data variables using a shared grid. When subsetting with an AOI, the instance of this class is cached to improve performance. The successive calls to `CFVariable$subset()` should use the same object returned from a single call to this function for this caching to work properly.

Value

The return value of the function is an `R6` object which uses reference semantics. Making changes to the returned object will be visible in all copies made of the object.

Examples

```
(aoi <- aoi(20, 60, -40, -20, 0.5))
```

`CFArray` *Array data extracted from a CF data variable*

Description

This class holds the data that is extracted from a CFVariable using the `data()`, `subset()` or `profile()` method. The instance of this class will additionally have the axes and other relevant information such as its attributes (as well as those of the axes) and the coordinate reference system.

Otherwise, a `CFArray` is detached from the data set where it was derived from. It is self-contained in the sense that all its constituent parts (axes, bounds, attributes, etc) are available and directly linked to the instance. For performance reasons, axes and their parts (e.g. bounds) are shared between instances of `CFArray` and `CFVariable`.

The class has a number of utility functions to extract the data in specific formats:

- `raw()`: The data without any further processing. The axes are as they are stored in the netCDF resource; there is thus no guarantee as to how the data is organized in the array. Dimnames will be set.
- `array()`: An array of the data which is organized as a standard R array with the axes of the data permuted to Y-X-others and Y-values in decreasing order. Dimnames will be set.
- `terra()`: The data is returned as a `terra::SpatRaster` (3D) or `terra::SpatRasterDataset` (4D) object, with all relevant structural metadata set. Package `terra` must be installed for this to work.
- `data.table()`: The data is returned as a `data.table`, with all data points on individual rows. Metadata is not maintained. Package `data.table` must be installed for this to work.

  The temporal axis of the data, if present, may be summarised using the `summarise()` method. The data is returned as a new `CFArray` instance.

  In general, the metadata from the netCDF resource will be lost when exporting to a different format insofar as those metadata are not recognized by the different format.

Super classes

ncdfCF::CFObject -> ncdfCF::CFVariableBase -> CFArray

Active bindings

dimnames (read-only) Retrieve dimnames of the data object.

Methods

Public methods:

- CFArray$new()
- CFArray$print()
- CFArray$raw()
- CFArray$array()
- CFArray$append()
- CFArray$terra()
- CFArray$data.table()
- CFArray$save()
- CFArray$clone()

Method new(): Create an instance of this class.

Usage:

```
CFArray$new(name, group, values, values_type, axes, crs, attributes)
```

Arguments:

name The name of the object.

group The group that this data should live in. This is usually an in-memory group, but it could be a regular group if the data is prepared for writing into a new netCDF file.

values The data of this object. The structure of the data depends on the method that produced it.

values_type The unpacked netCDF data type for this object.

axes A list of CFAxis descendant instances that describe the axes of the argument values.

crs The CFGridMapping instance of this data object, or NULL when no grid mapping is available.

attributes A data.frame with the attributes associated with the data in argument values.

Returns: An instance of this class.

Method print(): Print a summary of the data object to the console.

Usage:

```
CFArray$print(...)
```

Arguments:

... Arguments passed on to other functions. Of particular interest is width = to indicate a maximum width of attribute columns.

Method raw(): Retrieve the data in the object exactly as it was produced by the operation on CFVariable.

Usage:

```
CFArray$raw()
```

Returns: The data in the object. This is usually an `array` with the contents along axes varying.

Method `array()`**:** Retrieve the data in the object in the form of an R array, with axis ordering Y-X-others and Y values going from the top down.

Usage:

```
CFArray$array()
```

Returns: An `array` of data in R ordering.

Method `append()`**:** Append the data from another `CFArray` instance to the current instance, along one of the axes. The operation will only succeed if the axes other than the one to append along have the same coordinates and the coordinates of the axis to append along have to be monotonically increasing or decreasing after appending.

Usage:

```
CFArray$append(from, along)
```

Arguments:

`from` The `CFArray` instance to append from.

`along` The name of the axis to append along. This must be a single character string and the named axis has to be present both in `self` and in the `CFArray` instance in argument `from`.

Returns: `self`, invisibly, with the arrays from `self` and `from` appended.

Method `terra()`**:** Convert the data to a `terra::SpatRaster` (3D) or a `terra::SpatRasterDataset` (4D) object. The data will be oriented to North-up. The 3rd dimension in the data will become layers in the resulting `SpatRaster`, any 4th dimension the data sets. The `terra` package needs to be installed for this method to work.

Usage:

```
CFArray$terra()
```

Returns: A `terra::SpatRaster` or `terra::SpatRasterDataset` instance.

Method `data.table()`**:** Retrieve the data in the object in the form of a `data.table`. The `data.table` package needs to be installed for this method to work.

Usage:

```
CFArray$data.table(var_as_column = FALSE)
```

Arguments:

`var_as_column` Logical to flag if the name of the variable should become a column (`TRUE`) or be used as the name of the column with the data values (`FALSE`, default). Including the name of the variable as a column is useful when multiple `data.table`s are merged into one.

Returns: A `data.table` with all data points in individual rows. All axes will become columns. Two attributes are added: `name` indicates the long name of this data variable, `units` indicates the physical unit of the data values.

Method `save()`**:** Save the data object to a netCDF file.

Usage:

```
CFArray$save(fn, pack = FALSE)
```

Arguments:

`fn` The name of the netCDF file to create.

`pack` Logical to indicate if the data should be packed. Packing is only useful for numeric data; packing is not performed on integer values. Packing is always to the "NC_SHORT" data type, i.e. 16-bits per value.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFArray$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFAuxiliaryLongLat` *CF auxiliary longitude-latitude variable*

Description

This class represents the longitude and latitude variables that compose auxiliary coordinate variable axes for X-Y grids that are not longitude-latitude.

The class provides access to the data arrays for longitude and latitude from the netCDF resource, as well as all the details that have been associated with both axes. Additionally, this class can generate the index to extract values on a long-lat grid of the associated X-Y grid data variable using a user-selectable extent and resolution.

Auxiliary longitude-latitude grids are only supported for reading from a netCDF resource. Creating an instance manually therefore has no practical purpose.

Super class

`ncdfCF::CFObject` -> `CFAuxiliaryLongLat`

Public fields

`varLong` The NCVariable instance of the longitude values.

`varLat` The NCVariable instance of the latitude values.

`boundsLong` The CFBounds instance for the longitude values of the grid.

`boundsLat` The CFBounds instance for the latitude values of the grid.

`axis_order` Either `c("X", "Y")` (default) or `c("Y", "X")` to indicate the orientation of the latitude and longitude grids.

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`name` (read-only) The name of the auxiliary lon-lat grid.

`aoi` Set or retrieve the AOI for the long-lat grid.

`lon` (read-only) Retrieve the longitude grid.

`lat` (read-only) Retrieve the latitude grid.

`extent` (read-only) Retrieve the extent of the longitude and latitude grids, including bounds if they have been set. The extent is reported as a numeric vector of the four elements minumum and maximum longitude and minimum and maximum latitude.

`dim` (read-only) The dimensions of the longitude and latitude grids.

`dimids` (read-only) The dimids of the longitude and latitude grids.

Methods

Public methods:

- `CFAuxiliaryLongLat$new()`
- `CFAuxiliaryLongLat$print()`
- `CFAuxiliaryLongLat$brief()`
- `CFAuxiliaryLongLat$sample_index()`
- `CFAuxiliaryLongLat$grid_index()`
- `CFAuxiliaryLongLat$clear_cache()`
- `CFAuxiliaryLongLat$clone()`

Method `new()`**:** Creating a new instance. It should normally not be useful to create an instance of this class other than upon reading a netCDF resource.

Usage:

```
CFAuxiliaryLongLat$new(varLong, varLat, boundsLong, boundsLat)
```

Arguments:

`varLong`, `varLat` The NCVariable instances with the longitude and latitude grid values, respectively.

`boundsLong`, `boundsLat` The bounds of the grid cells for the longitude and latitude, respectively, if set.

Method `print()`**:** Summary of the auxiliary longitude-latitude variable printed to the console.

Usage:

```
CFAuxiliaryLongLat$print()
```

Method `brief()`**:** Some details of the auxiliary longitude-latitude grid.

Usage:

```
CFAuxiliaryLongLat$brief()
```

Returns: A 2-row `data.frame` with some details of the grid components.

Method `sample_index()`**:** Return the indexes into the X (longitude) and Y (latitude) axes of the original data grid of the points closest to the supplied longitudes and latitudes, up to a maximum distance.

Usage:

```
CFAuxiliaryLongLat$sample_index(x, y, maxDist = NULL)
```

Arguments:

`x, y` Vectors of longitude and latitude values in decimal degrees, respectively.

`maxDist` Numeric value in decimal degrees of the maximum distance between the sampling point and the closest grid cell. If omitted (default), the distance is calculated from the nominal resolution of the grids.

Returns: A matrix with two columns `X` and `Y` and as many rows as arguments `x` and `y`. The `X` and `Y` columns give the index into the grid of the sampling points, or `c(NA, NA)` is no grid point is located within the `maxDist` distance from the sampling point.

Method `grid_index()`**:** Compute the indices for the AOI into the data grid.

Usage:

```
CFAuxiliaryLongLat$grid_index()
```

Returns: An integer matrix with the dimensions of the AOI, where each grid cell gives the linear index value into the longitude and latitude grids.

Method `clear_cache()`**:** Clears the cache of pre-computed grid index values if an AOI has been set.

Usage:

```
CFAuxiliaryLongLat$clear_cache(full = FALSE)
```

Arguments:

`full` Logical (default = `FALSE`) that indicates if longitude and latitude grid arrays should be cleared as well to save space. These will then be re-read from file if a new AOI is set.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFAuxiliaryLongLat$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFAxis` *CF axis object*

Description

This class is a basic ancestor to all classes that represent CF axes. More useful classes use this class as ancestor.

This super-class does manage the "coordinates" of the axis, i.e. the values along the axis. This could be the values of the axis as stored on file, but it can also be the values from an auxiliary coordinate set, in the form of a CFLabel instance. The coordinate set to use in display, selection and processing is selectable through methods and fields in this class.

Super class

ncdfCF::CFObject -> CFAxis

Public fields

NCdim The NCDimension that stores the netCDF dimension details.

orientation A character "X", "Y", "Z" or "T" to indicate the orientation of the axis, or an empty string if not known or different.

bounds The boundary values of this axis, if set.

Active bindings

friendlyClassName (read-only) A nice description of the class.

dimid (read-only) The netCDF dimension id of this axis.

length (read-only) The declared length of this axis.

values (read-only) Retrieve the raw values of the axis. In general you should use the coordinates field rather than this one.

coordinates (read-only) Retrieve the coordinate values of the active coordinate set from the axis.

auxiliary Set or retrieve auxiliary coordinates for the axis. On assignment, the value must be an instance of CFLabel or a CFAxis descendant, which is added to the end of the list of coordinate sets. On retrieval, the active CFLabel or CFAxis instance or NULL when the active coordinate set is the primary axis coordinates.

coordinate_names Retrieve the names of the coordinate sets defined for the axis, as a character vector. The first element in the vector is the name of the axis and it refers to the values of the coordinates as stored in the netCDF file. Following elements refer to auxiliary coordinates.

active_coordinates Set or retrieve the name of the coordinate set to use with the axis for printing to the console as well as for processing methods such as subset().

unlimited (read-only) Logical to indicate if the axis has an unlimited dimension.

Methods

Public methods:

- CFAxis$new()
- CFAxis$print()
- CFAxis$brief()
- CFAxis$shard()
- CFAxis$peek()
- CFAxis$time()
- CFAxis$identical()
- CFAxis$can_append()
- CFAxis$subset()
- CFAxis$indexOf()
- CFAxis$write()

- CFAxis$clone()

Method new(): Create a new CF axis instance from a dimension and a variable in a netCDF resource. This method is called upon opening a netCDF resource by the initialize() method of a descendant class suitable for the type of axis.

Creating a new axis is more easily done with the makeAxis() function.

Usage:

```
CFAxis$new(nc_var, nc_dim, orientation)
```

Arguments:

nc_var The NCVariable instance upon which this CF axis is based.

nc_dim The NCDimension instance upon which this CF axis is based.

orientation The orientation of the axis: "X", "Y", "Z" "T", or "" when not known or relevant.

Returns: A basic CFAxis object.

Method print(): Prints a summary of the axis to the console. This method is typically called by the print() method of descendant classes.

Usage:

```
CFAxis$print(...)
```

Arguments:

... Arguments passed on to other functions. Of particular interest is width = to indicate a maximum width of attribute columns.

Returns: self, invisibly.

Method brief(): Some details of the axis.

Usage:

```
CFAxis$brief()
```

Returns: A 1-row data.frame with some details of the axis.

Method shard(): Very concise information on the axis. The information returned by this function is very concise and most useful when combined with similar information from other axes.

Usage:

```
CFAxis$shard()
```

Returns: Character string with very basic axis information.

Method peek(): Retrieve interesting details of the axis.

Usage:

```
CFAxis$peek(with_groups = TRUE)
```

Arguments:

with_groups Should group information be included? The safe option is TRUE (default) when the netCDF resource has groups because names may be duplicated among objects in different groups.

Returns: A 1-row data.frame with details of the axis.

Method `time()`**:** Return the `CFTime` instance that represents time. This method is only useful for `CFAxisTime` instances having time information. This stub is here to make the call to this method succeed with no result for the other descendant classes.

Usage:

```
CFAxis$time()
```

Returns: `NULL`

Method `identical()`**:** Tests if the axis passed to this method is identical to `self`. This only tests for generic properties - class, length and name

- with further assessment done in sub-classes.

Usage:

```
CFAxis$identical(axis)
```

Arguments:

`axis` The `CFAxis` instance to test.

Returns: `TRUE` if the two axes are identical, `FALSE` if not.

Method `can_append()`**:** Tests if the axis passed to this method can be appended to `self`. This only tests for generic properties - class, mode of the values and name - with further assessment done in sub-classes.

Usage:

```
CFAxis$can_append(axis)
```

Arguments:

`axis` The `CFAxis` descendant instance to test.

Returns: `TRUE` if the passed axis can be appended to `self`, `FALSE` if not.

Method `subset()`**:** Return an axis spanning a smaller coordinate range. This method is "virtual" in the sense that it does not do anything other than return `NULL`. This stub is here to make the call to this method succeed with no result for the `CFAxis` descendants that do not implement this method.

Usage:

```
CFAxis$subset(group, rng = NULL)
```

Arguments:

`group` The group to create the new axis in.

`rng` The range of indices whose values from this axis to include in the returned axis. If the value of the argument is `NULL`, return the entire axis.

Returns: `NULL`

Method `indexOf()`**:** Find indices in the axis domain. Given a vector of numerical, timestamp or categorical coordinates `x`, find their indices in the coordinates of the axis.

This is a virtual method. For more detail, see the corresponding method in descendant classes.

Usage:

```
CFAxis$indexOf(x, method = "constant", rightmost.closed = TRUE)
```

Arguments:

`x` Vector of numeric, timestamp or categorial coordinates to find axis indices for. The timestamps can be either character, POSIXct or Date vectors. The type of the vector has to correspond to the type of the axis values.

`method` Single character value of "constant" or "linear".

`rightmost.closed` Whether or not to include the upper limit. Default is `TRUE`.

Returns: Numeric vector of the same length as `x`.

Method `write()`**:** Write the axis to a netCDF file, including its attributes.

Usage:

```
CFAxis$write(nc = NULL)
```

Arguments:

`nc` The handle of the netCDF file opened for writing or a group in the netCDF file. If `NULL`, write to the file or group where the axis was read from (the file must have been opened for writing). If not `NULL`, the handle to a netCDF file or a group therein.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFAxis$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFAxisCharacter` *CF character axis object*

Description

This class represent CF axes that use categorical character labels as coordinate values. Note that this is different from a CFLabel, which is associated with an axis but not an axis itself.

This is an extension to the CF Metadata Conventions. As per CF, axes are required to have numerical values, which is relaxed here.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFAxis` -> `CFAxisCharacter`

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`dimnames` (read-only) The coordinates of the axis as a character vector.

Methods

Public methods:

- CFAxisCharacter$new()
- CFAxisCharacter$brief()
- CFAxisCharacter$identical()
- CFAxisCharacter$append()
- CFAxisCharacter$indexOf()
- CFAxisCharacter$clone()

Method `new()`**:** Create a new instance of this class.
Creating a new character axis is more easily done with the `makeAxis()` function.

Usage:

```
CFAxisCharacter$new(nc_var, nc_dim, orientation, values)
```

Arguments:

`nc_var` The netCDF variable that describes this instance.

`nc_dim` The netCDF dimension that describes the dimensionality.

`orientation` The orientation (`X`, `Y`, `Z`, or `T`) or `""` if different or unknown.

`values` The character coordinates of this axis.

Method `brief()`**:** Some details of the axis.

Usage:

```
CFAxisCharacter$brief()
```

Returns: A 1-row `data.frame` with some details of the axis.

Method `identical()`**:** Tests if the axis passed to this method is identical to `self`.

Usage:

```
CFAxisCharacter$identical(axis)
```

Arguments:

`axis` The `CFAxisCharacter` instance to test.

Returns: `TRUE` if the two axes are identical, `FALSE` if not.

Method `append()`**:** Append a vector of values at the end of the current values of the axis.

Usage:

```
CFAxisCharacter$append(from)
```

Arguments:

`from` An instance of `CFAxisCharacter` whose values to append to the values of `self`.

Returns: A new `CFAxisCharacter` instance with values from `self` and the `from` axis appended.

Method `indexOf()`**:** Find indices in the axis domain. Given a vector of character strings `x`, find their indices in the coordinates of the axis.

Usage:

```
CFAxisCharacter$indexOf(x, method = "constant", rightmost.closed = TRUE)
```

Arguments:

`x` Vector of character strings to find axis indices for.

`method` Ignored.

`rightmost.closed` Ignored.

Returns: Numeric vector of the same length as `x`. Values of `x` that are not equal to a coordinate of the axis are returned as `NA`.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFAxisCharacter$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFAxisDiscrete` *CF discrete axis object*

Description

This class represent discrete CF axes, i.e. those axes whose coordinate values do not represent a physical property. The coordinate values are ordinal values equal to the index into the axis.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFAxis` -> `CFAxisDiscrete`

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`dimnames` (read-only) The coordinates of the axis as an integer vector, or labels for every axis element if they have been set.

Methods

Public methods:

- `CFAxisDiscrete$new()`
- `CFAxisDiscrete$print()`
- `CFAxisDiscrete$brief()`
- `CFAxisDiscrete$append()`
- `CFAxisDiscrete$indexOf()`
- `CFAxisDiscrete$subset()`
- `CFAxisDiscrete$write()`
- `CFAxisDiscrete$clone()`

Method `new()`**:** Create a new instance of this class.
Creating a new discrete axis is more easily done with the `makeDiscreteAxis()` function.

Usage:

```
CFAxisDiscrete$new(nc_var, nc_dim, orientation, dim_only = FALSE)
```

Arguments:

`nc_var` The netCDF variable that describes this instance.

`nc_dim` The netCDF dimension that describes the dimensionality.

`orientation` The orientation (`X`, `Y`, `Z`, or `T`) or `""` if different or unknown.

`dim_only` Flag if this axis only has a dimension on file but no NC variable.

Method `print()`**:** Summary of the axis printed to the console.

Usage:

```
CFAxisDiscrete$print(...)
```

Arguments:

`...` Arguments passed on to other functions. Of particular interest is `width =` to indicate a maximum width of attribute columns.

Returns: `self`, invisibly.

Method `brief()`**:** Some details of the axis.

Usage:

```
CFAxisDiscrete$brief()
```

Returns: A 1-row `data.frame` with some details of the axis.

Method `append()`**:** Append a vector of values at the end of the current values of the axis. In a discrete axis the values are always a simple sequence so the appended values extend the sequence, rather than using the values from axis `from`.

Usage:

```
CFAxisDiscrete$append(from)
```

Arguments:

`from` An instance of `CFAxisDiscrete` whose length to add to the length of `self`.

Returns: A new `CFAxisDiscrete` with the combined length of `self` and the `from` axis.

Method `indexOf()`**:** Find indices in the axis domain. Given a vector of numerical values `x`, find their indices in the values of the axis. In effect, this returns index values into the axis, but outside values will be dropped.

Usage:

```
CFAxisDiscrete$indexOf(x, method = "constant", rightmost.closed = TRUE)
```

Arguments:

`x` Vector of numeric values to find axis indices for.

`method` Ignored.

`rightmost.closed` Ignored.

Returns: Numeric vector of the same length as `x`. Values of `x` outside of the range of the values in the axis are returned as `NA`.

Method `subset()`**:** Return an axis spanning a smaller coordinate range. This method returns an axis which spans the range of indices given by the `rng` argument.

Usage:

```
CFAxisDiscrete$subset(group, rng = NULL)
```

Arguments:

`group` The group to create the new axis in.

`rng` The range of indices from this axis to include in the returned axis.

Returns: A `CFAxisDiscrete` instance covering the indicated range of indices. If the value of the argument is `NULL`, return the entire axis.

Method `write()`**:** Write the axis to a netCDF file, including its attributes, but only if it has an associated NC variable in the file.

Usage:

```
CFAxisDiscrete$write(nc = NULL)
```

Arguments:

`nc` The handle of the netCDF file opened for writing or a group in the netCDF file. If `NULL`, write to the file or group where the axis was read from (the file must have been opened for writing). If not `NULL`, the handle to a netCDF file or a group therein.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFAxisDiscrete$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFAxisLatitude` *Latitude CF axis object*

Description

This class represents a latitude axis. Its values are numeric. This class adds some logic that is specific to latitudes, such as their range, orientation and meaning.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFAxis` -> `ncdfCF::CFAxisNumeric` -> `CFAxisLatitude`

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

Methods

Public methods:

- `CFAxisLatitude$new()`
- `CFAxisLatitude$subset()`
- `CFAxisLatitude$clone()`

Method `new()`**:** Create a new instance of this class.

Creating a new latitude axis is more easily done with the `makeLatitudeAxis()` function.

Usage:

```
CFAxisLatitude$new(nc_var, nc_dim, values)
```

Arguments:

`nc_var` The netCDF variable that describes this instance.

`nc_dim` The netCDF dimension that describes the dimensionality.

`values` The coordinates of this axis.

Method `subset()`**:** Return an axis spanning a smaller coordinate range. This method returns an axis which spans the range of indices given by the `rng` argument.

Usage:

```
CFAxisLatitude$subset(group, rng = NULL)
```

Arguments:

`group` The group to create the new axis in.

`rng` The range of values from this axis to include in the returned axis.

Returns: A `CFAxisLatitude` instance covering the indicated range of indices. If the value of the argument is `NULL`, return the entire axis.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFAxisLatitude$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFAxisLongitude` *Longitude CF axis object*

Description

This class represents a longitude axis. Its values are numeric. This class is used for axes that represent longitudes. This class adds some logic that is specific to longitudes, such as their range, orientation and their meaning. (In the near future, it will also support selecting data that crosses the 0-360 degree boundary.)

Super classes

ncdfCF::CFObject -> ncdfCF::CFAxis -> ncdfCF::CFAxisNumeric -> CFAxisLongitude

Active bindings

friendlyClassName (read-only) A nice description of the class.

Methods

Public methods:

- CFAxisLongitude$new()
- CFAxisLongitude$subset()
- CFAxisLongitude$clone()

Method new(): Create a new instance of this class.

Creating a new longitude axis is more easily done with the makeLongitudeAxis() function.

Usage:

```
CFAxisLongitude$new(nc_var, nc_dim, values)
```

Arguments:

nc_var The netCDF variable that describes this instance.

nc_dim The netCDF dimension that describes the dimensionality.

values The coordinates of this axis.

Method subset(): Return an axis spanning a smaller coordinate range. This method returns an axis which spans the range of indices given by the rng argument.

Usage:

```
CFAxisLongitude$subset(group, rng = NULL)
```

Arguments:

group The group to create the new axis in.

rng The range of values from this axis to include in the returned axis.

Returns: A CFAxisLongitude instance covering the indicated range of indices. If the value of the argument is NULL, return the entire axis.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
CFAxisLongitude$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

CFAxisNumeric *Numeric CF axis object*

Description

This class represents a numeric axis. Its values are numeric. This class is used for axes with numeric values but without further knowledge of their nature. More specific classes descend from this class.

Super classes

ncdfCF::CFObject -> ncdfCF::CFAxis -> CFAxisNumeric

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`dimnames` (read-only) The coordinates of the axis as a vector. These are by default the values of the axis, but it could also be a set of auxiliary coordinates, if they have been set.

Methods

Public methods:

- CFAxisNumeric$new()
- CFAxisNumeric$print()
- CFAxisNumeric$brief()
- CFAxisNumeric$range()
- CFAxisNumeric$indexOf()
- CFAxisNumeric$identical()
- CFAxisNumeric$append()
- CFAxisNumeric$subset()
- CFAxisNumeric$clone()

Method `new()`**:** Create a new instance of this class.

Creating a new axis is more easily done with the `makeAxis()` function.

Usage:

```
CFAxisNumeric$new(nc_var, nc_dim, orientation, values)
```

Arguments:

`nc_var` The netCDF variable that describes this instance.

`nc_dim` The netCDF dimension that describes the dimensionality.

`orientation` The orientation (`X`, `Y`, `Z`, or `T`) or `""` if different or unknown.

`values` The coordinates of this axis.

Method `print()`**:** Summary of the axis printed to the console.

Usage:

```
CFAxisNumeric$print(...)
```

Arguments:

`...` Arguments passed on to other functions. Of particular interest is `width =` to indicate a maximum width of attribute columns.

Returns: `self`, invisibly.

Method `brief()`**:** Some details of the axis.

Usage:

```
CFAxisNumeric$brief()
```

Returns: A 1-row `data.frame` with some details of the axis.

Method `range()`**:** Retrieve the range of coordinate values in the axis.

Usage:

```
CFAxisNumeric$range()
```

Returns: A numeric vector with two elements with the minimum and maximum values in the axis, respectively.

Method `indexOf()`**:** Retrieve the indices of supplied coordinates on the axis. If the axis has bounds then the supplied coordinates must fall within the bounds to be considered valid.

Usage:

```
CFAxisNumeric$indexOf(x, method = "constant", rightmost.closed = TRUE)
```

Arguments:

`x` A numeric vector of coordinates whose indices into the axis to extract.

`method` Extract index values without ("constant", the default) or with ("linear") fractional parts.

`rightmost.closed` Whether or not to include the upper limit. This parameter is ignored for this class, effectively it always is `TRUE`.

Returns: A vector giving the indices in `x` of valid coordinates provided. Values of `x` outside of the range of the coordinates in the axis are returned as `NA`.

Method `identical()`**:** Tests if the axis passed to this method is identical to `self`.

Usage:

```
CFAxisNumeric$identical(axis)
```

Arguments:

`axis` The `CFAxisNumeric` or sub-class instance to test.

Returns: `TRUE` if the two axes are identical, `FALSE` if not.

Method `append()`**:** Append a vector of values at the end of the current values of the axis.

Usage:

```
CFAxisNumeric$append(from)
```

Arguments:

`from` An instance of `CFAxisNumeric` or any of its descendants whose values to append to the values of `self`.

Returns: A new `CFAxisNumeric` or descendant instance with values from self and the `from` axis appended.

Method `subset()`**:** Return an axis spanning a smaller coordinate range. This method returns an axis which spans the range of indices given by the `rng` argument.

Usage:

```
CFAxisNumeric$subset(group, rng = NULL)
```

Arguments:

`group` The group to create the new axis in.

`rng` The range of indices whose values from this axis to include in the returned axis.

Returns: A `CFAxisNumeric` instance covering the indicated range of indices. If the value of the argument is `NULL`, return the entire axis.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFAxisNumeric$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFAxisTime` *Time axis object*

Description

This class represents a time axis. The functionality is provided by the `CFTime` class in the `CFtime` package.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFAxis` -> `CFAxisTime`

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`dimnames` (read-only) The coordinates of the axis as a character vector.

Methods

Public methods:

- `CFAxisTime$new()`
- `CFAxisTime$print()`
- `CFAxisTime$brief()`
- `CFAxisTime$time()`
- `CFAxisTime$identical()`

- CFAxisTime$append()
- CFAxisTime$indexOf()
- CFAxisTime$slice()
- CFAxisTime$subset()
- CFAxisTime$write()
- CFAxisTime$clone()

Method `new()`**:** Create a new instance of this class.
Creating a new time axis is more easily done with the `makeTimeAxis()` function.

Usage:

```
CFAxisTime$new(nc_var, nc_dim, values)
```

Arguments:

`nc_var` The netCDF variable that describes this instance.

`nc_dim` The netCDF dimension that describes the dimensionality.

`values` The `CFTime` instance that manages this axis.

Method `print()`**:** Summary of the time axis printed to the console.

Usage:

```
CFAxisTime$print(...)
```

Arguments:

`...` Arguments passed on to other functions. Of particular interest is `width =` to indicate a maximum width of attribute columns.

Returns: `self`, invisibly.

Method `brief()`**:** Some details of the axis.

Usage:

```
CFAxisTime$brief()
```

Returns: A 1-row `data.frame` with some details of the axis.

Method `time()`**:** Retrieve the `CFTime` instance that manages the values of this axis.

Usage:

```
CFAxisTime$time()
```

Returns: An instance of `CFTime`.

Method `identical()`**:** Tests if the axis passed to this method is identical to `self`.

Usage:

```
CFAxisTime$identical(axis)
```

Arguments:

`axis` The `CFAxisTime` instance to test.

Returns: `TRUE` if the two axes are identical, `FALSE` if not.

Method `append()`**:** Append a vector of time values at the end of the current values of the axis.

Usage:

```
CFAxisTime$append(from)
```

Arguments:

`from` An instance of `CFAxisTime` whose values to append to the values of `self`.

Returns: A new `CFAxisTime` instance with values from self and the `from` axis appended.

Method `indexOf()`**:** Retrieve the indices of supplied values on the time axis.

Usage:

```
CFAxisTime$indexOf(x, method = "constant", rightmost.closed = FALSE)
```

Arguments:

`x` A vector of timestamps whose indices into the time axis to extract.

`method` Extract index values without ("constant", the default) or with ("linear") fractional parts.

`rightmost.closed` Whether or not to include the upper limit. Default is `FALSE`.

Returns: An integer vector giving the indices in the time axis of valid values in `x`, or `NA` if the value is not valid.

Method `slice()`**:** Retrieve the indices of the time axis falling between two extreme values.

Usage:

```
CFAxisTime$slice(x, rightmost.closed = FALSE)
```

Arguments:

`x` A vector of two timestamps in between of which all indices into the time axis to extract.

`rightmost.closed` Whether or not to include the upper limit. Default is `FALSE`.

Returns: An integer vector giving the indices in the time axis between values in `x`, or `integer(0)` if none of the values are valid.

Method `subset()`**:** Return an axis spanning a smaller coordinate range. This method returns an axis which spans the range of indices given by the `rng` argument.

Usage:

```
CFAxisTime$subset(group, rng = NULL)
```

Arguments:

`group` The group to create the new axis in.

`rng` The range of indices whose values from this axis to include in the returned axis.

Returns: A `CFAxisTime` instance covering the indicated range of indices. If the value of the argument is `NULL`, return the entire axis.

Method `write()`**:** Write the axis to a netCDF file, including its attributes. If the calendar name is "gregorian", it will be set to the functionally identical calendar "standard" as the former is deprecated.

Usage:

```
CFAxisTime$write(nc = NULL)
```

Arguments:

`nc` The handle of the netCDF file opened for writing or a group in the netCDF file. If `NULL`, write to the file or group where the axis was read from (the file must have been opened for writing). If not `NULL`, the handle to a netCDF file or a group therein.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFAxisTime$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFAxisVertical` *Parametric vertical CF axis object*

Description

This class represents a parametric vertical axis. It is defined through an index value that is contained in the axis, with additional NCVariable instances that hold ancillary data with which to calculate dimensional axis values. It is used in atmosphere and ocean data sets. Non-parametric vertical axes are stored in an CFAxisNumeric instance.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFAxis` -> `ncdfCF::CFAxisNumeric` -> `CFAxisVertical`

Public fields

`parameter_name` The 'standard_name' attribute of the NCVariable that identifies the parametric form of this axis.

`computed_name` The standard name for the computed values of the axis.

`computed_units` The unit of the computed values of the axis.

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`formula_terms` A `data.frame` with the "formula_terms" to calculate the parametric axis values.

`dimnames` (read-only) The coordinates of the axis.

Methods

Public methods:

- `CFAxisVertical$new()`
- `CFAxisVertical$clone()`

Method `new()`**:** Create a new instance of this class.

Usage:

```
CFAxisVertical$new(nc_var, nc_dim, values, standard_name)
```

Arguments:

`nc_var` The netCDF variable that describes this instance.

`nc_dim` The netCDF dimension that describes the dimensionality.

`values` The coordinates of this axis.

`standard_name` Character string with the "standard_name" that defines the meaning, and processing of coordinates, of this axis.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFAxisVertical$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://cfconventions.org/Data/cf-conventions/cf-conventions-1.12/cf-conventions.html#parametric-vertical-coordinate

---

`CFBounds` *CF bounds variable*

Description

This class represents the bounds of an axis or an auxiliary longitude-latitude grid.

The class manages the bounds information for an axis (2 vertices per element) or an auxiliary longitude-latitude grid (4 vertices per element).

Super class

`ncdfCF::CFObject` -> `CFBounds`

Public fields

`NCdim` The NCDimension that stores the netCDF dimension details of the bounds dimension (as opposed to the dimension of the associated axis).

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`coordinates` (read-only) Retrieve the boundary values.

Methods

Public methods:

- CFBounds$new()
- CFBounds$print()
- CFBounds$range()
- CFBounds$sub_bounds()
- CFBounds$write()
- CFBounds$clone()

Method `new()`**:** Create an instance of this class.

Usage:

```
CFBounds$new(nc_var, nc_dim, values)
```

Arguments:

`nc_var` The NC variable that describes this instance.

`nc_dim` The NC dimension that defines the vertices of the bounds.

`values` A matrix with the bounds values.

Method `print()`**:** Print a summary of the object to the console.

Usage:

```
CFBounds$print(...)
```

Arguments:

`...` Arguments passed on to other functions. Of particular interest is `width =` to indicate a maximum width of attribute columns.

Method `range()`**:** Retrieve the lowest and highest value in the bounds.

Usage:

```
CFBounds$range()
```

Method `sub_bounds()`**:** Return bounds spanning a smaller coordinate range.
This method returns bounds which spans the range of indices given by the `rng` argument.

Usage:

```
CFBounds$sub_bounds(group, rng)
```

Arguments:

`group` The group to create the new bounds in.

`rng` The range of values from this bounds object to include in the returned object.

Returns: A `CFBounds` instance covering the indicated range of indices.

Method `write()`**:** Write the bounds variable to a netCDF file. This method should not be called directly; instead, `CFArray::save()` will call this method automatically.

Usage:

```
CFBounds$write(h, object_name)
```

Arguments:

`h` The handle to a netCDF file open for writing.

`object_name` The name of the object that uses these bounds, usually an axis but could also be an auxiliary CV or a parametric Z axis.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFBounds$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFCellMeasure` *CF cell measure variable*

Description

This class represents a CF cell measure variable, the object that indicates the area or volume of every grid cell in referencing data variables.

If a cell measure variable is external to the current file, an instance will still be created for it, but the user must link the external file to this instance before it can be used in analysis.

Public fields

`group` The NCGroup that this object is located in.

`measure` The measure of this instance. Either "area" or "volume".

`name` The name of this instance, which must refer to a NC variable or an external variable.

Methods

Public methods:

- `CFCellMeasure$new()`
- `CFCellMeasure$print()`
- `CFCellMeasure$data()`
- `CFCellMeasure$register()`
- `CFCellMeasure$link()`
- `CFCellMeasure$clone()`

Method `new()`**:** Create an instance of this class.

Usage:

```
CFCellMeasure$new(grp, measure, name = NULL, nc_var = NULL, axes = NULL)
```

Arguments:

`grp` The group that this CF cell measure variable lives in.

`measure` The measure of this object. Must be either of "area" or "volume".

`name` The name of the cell measure variable. May be omitted if argument `nc_var` is specified.

`nc_var` The netCDF variable that defines this CF cell measure object. `NULL` for an external variable.

`axes` List of CFAxis instances that describe the dimensions of the cell measure object. `NULL` for an external variable.

Returns: An instance of this class.

Method `print()`**:** Print a summary of the cell measure variable to the console.

Usage:

```
CFCellMeasure$print(...)
```

Arguments:

`...` Arguments passed on to other functions. Of particular interest is `width =` to indicate a maximum width of attribute columns.

Method `data()`**:** Retrieve the values of the cell measure variable.

Usage:

```
CFCellMeasure$data()
```

Returns: The values of the cell measure as a CFArray instance.

Method `register()`**:** Register a CFVariable which is using this cell measure variable. A check is performed on the compatibility between the data variable and this cell measure variable.

Usage:

```
CFCellMeasure$register(var)
```

Arguments:

`var` A `CFVariable` instance to link to this instance.

Returns: Self, invisibly.

Method `link()`**:** Link the cell measure variable to an external netCDF resource. The resource will be opened and the appropriate data variable will be linked to this instance. If the axes or other properties of the external resource are not compatible with this instance, an error will be raised.

Usage:

```
CFCellMeasure$link(resource)
```

Arguments:

`resource` The name of the netCDF resource to open, either a local file name or a remote URI.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFCellMeasure$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFDataset` *CF data set*

Description

This class represents a CF data set, the object that encapsulates a netCDF resource. You should never have to instantiate this class directly; instead, call `open_ncdf()` which will return an instance that has all properties read from the netCDF resource. Class methods can then be called, or the base R functions called with this instance.

The CF data set instance provides access to all the objects in the netCDF resource, organized in groups.

Public fields

`name` The name of the netCDF resource. This is extracted from the URI (file name or URL).

`keep_open` Logical flag to indicate if the netCDF resource has to remain open after reading the metadata. This should be enabled typically only for programmatic access or when a remote resource has an expensive access protocol (i.e. 2FA). The resource has to be explicitly closed with `close()` after use. Note that when a data set is opened with `keep_open = TRUE` the resource may still be closed by the operating system or the remote server.

`root` Root of the group hierarchy through which all elements of the netCDF resource are accessed. It is **strongly discouraged** to manipulate the objects in the group hierarchy directly. Use the provided access methods instead.

`file_type` The type of data in the netCDF resource, if identifiable. In terms of the CF Metadata Conventions, this includes discrete sampling geometries (DSG). Other file types that can be identified include L3b files used by NASA and NOAA for satellite imagery (these data sets need special processing), and CMIP5, CMIP6 and CORDEX climate projection data.

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`resource` (read-only) The connection details of the netCDF resource. This is for internal use only.

`uri` (read-only) The connection string to the netCDF resource.

`conventions` (read-only) Returns the conventions that this netCDF resource conforms to.

`var_names` (read-only) Vector of names of variables in this data set.

`axis_names` (read-only) Vector of names of axes in this data set.

Methods

Public methods:

- `CFDataset$new()`
- `CFDataset$print()`
- `CFDataset$hierarchy()`
- `CFDataset$objects_by_standard_name()`

- CFDataset$has_subgroups()
- CFDataset$find_by_name()
- CFDataset$variables()
- CFDataset$axes()
- CFDataset$attributes()
- CFDataset$attribute()
- CFDataset$clone()

Method `new()`**:** Create an instance of this class.

Usage:

```
CFDataset$new(name, resource, keep_open, format)
```

Arguments:

`name` The name that describes this instance.

`resource` An instance of `CFResource` that links to the netCDF resource.

`keep_open` Logical. Should the netCDF resource be kept open for further access?

`format` Character string with the format of the netCDF resource as reported by the call opening the resource.

Method `print()`**:** Summary of the data set printed to the console.

Usage:

```
CFDataset$print(...)
```

Arguments:

`...` Arguments passed on to other functions. Of particular interest is `width =` to indicate a maximum width of attribute columns.

Method `hierarchy()`**:** Print the group hierarchy to the console.

Usage:

```
CFDataset$hierarchy()
```

Method `objects_by_standard_name()`**:** Get objects by standard_name. Several conventions define standard vocabularies for physical properties. The standard names from those vocabularies are usually stored as the "standard_name" attribute with variables or axes. This method retrieves all variables or axes that list the specified "standard_name" in its attributes.

Usage:

```
CFDataset$objects_by_standard_name(standard_name)
```

Arguments:

`standard_name` Optional, a character string to search for a specific "standard_name" value in variables and axes.

Returns: If argument `standard_name` is provided, a character vector of variable or axis names. If argument `standard_name` is missing or an empty string, a named list with all "standard_name" attribute values in the the netCDF resource; each list item is named for the variable or axis.

Method `has_subgroups()`**:** Does the netCDF resource have subgroups? Newer versions of the `netcdf` library, specifically `netcdf4`, can organize dimensions and variables in groups. This method will report if the data set is indeed organized with subgroups.

Usage:

```
CFDataset$has_subgroups()
```

Returns: Logical to indicate that the netCDF resource uses subgroups.

Method `find_by_name()`**:** Find an object by its name. Given the name of a CF data variable or axis, possibly preceded by an absolute group path, return the object to the caller.

Usage:

```
CFDataset$find_by_name(name, scope = "CF")
```

Arguments:

`name` The name of a CF data variable or axis, with an optional absolute group path.

`scope` The scope to look for the name. Either "CF" (default) to search for CF variables or axes, or "NC" to look for groups or NC variables.

Returns: The object with the provided name. If the object is not found, returns `NULL`.

Method `variables()`**:** This method lists the CF data variables located in this netCDF resource, including those in subgroups.

Usage:

```
CFDataset$variables()
```

Returns: A list of `CFVariable` instances.

Method `axes()`**:** This method lists the axes located in this netCDF resource, including axes in subgroups.

Usage:

```
CFDataset$axes()
```

Returns: A list of `CFAxis` descendants.

Method `attributes()`**:** List all the attributes of a group. This method returns a `data.frame` containing all the attributes of the indicated `group`.

Usage:

```
CFDataset$attributes(group)
```

Arguments:

`group` The name of the group whose attributes to return. If the argument is missing, the global attributes will be returned.

Returns: A `data.frame` of attributes.

Method `attribute()`**:** Retrieve global attributes of the data set.

Usage:

```
CFDataset$attribute(att, field = "value")
```

Arguments:

`att` Vector of character strings of attributes to return.

`field` The field of the attribute to return values from. This must be "value" (default) or "type".

Returns: If the `field` argument is "type", a character string. If `field` is "value", a single value of the type of the attribute, or a vector when the attribute has multiple values. If no attribute is named with a value of argument `att` `NA` is returned.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFDataset$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFGridMapping` *CF grid mapping object*

Description

This class contains the details for a coordinate reference system, or grid mapping in CF terms, of a data variable.

When reporting the coordinate reference system to the caller, a character string in WKT2 format is returned, following the OGC standard.

Super class

`ncdfCF::CFObject` -> `CFGridMapping`

Public fields

`grid_mapping_name` The name of the grid mapping.

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

Methods

Public methods:

- `CFGridMapping$new()`
- `CFGridMapping$print()`
- `CFGridMapping$brief()`
- `CFGridMapping$wkt2()`
- `CFGridMapping$write()`
- `CFGridMapping$clone()`

Method `new()`**:** Create a new instance of this class.

Usage:

```
CFGridMapping$new(nc_var, name)
```

Arguments:

`nc_var` The netCDF variable that describes this instance.

`name` The formal grid mapping name from the attribute.

Method `print()`**:** Prints a summary of the grid mapping to the console.

Usage:

```
CFGridMapping$print()
```

Method `brief()`**:** Retrieve a 1-row `data.frame` with some information on this grid mapping.

Usage:

```
CFGridMapping$brief()
```

Method `wkt2()`**:** Retrieve the CRS string for a specific variable.

Usage:

```
CFGridMapping$wkt2(axis_info)
```

Arguments:

`axis_info` A list with information that describes the axes of the `CFVariable` or `CFArray` instance to describe.

Returns: A character string with the CRS in WKT2 format.

Method `write()`**:** Write the CRS object to a netCDF file.

Usage:

```
CFGridMapping$write(h)
```

Arguments:

`h` Handle to the netCDF file opened for writing.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFGridMapping$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://docs.ogc.org/is/18-010r11/18-010r11.pdf

CFLabel *CF label object*

Description

This class represent CF labels, i.e. an NC variable of character type that provides a textual label for a discrete or general numeric axis. See also CFAxisCharacter, which is an axis with character labels.

Super class

ncdfCF::CFObject -> CFLabel

Public fields

NCdim The NCDimension that stores the netCDF dimension details.

Active bindings

friendlyClassName (read-only) A nice description of the class.

coordinates (read-only) The label set as a vector.

length (read-only) The number of labels in the set.

dimid (read-only) The netCDF dimension id of this label set.

Methods

Public methods:

- CFLabel$new()
- CFLabel$print()
- CFLabel$subset()
- CFLabel$write()
- CFLabel$clone()

Method new(): Create a new instance of this class.

Usage:

```
CFLabel$new(nc_var, nc_dim, values)
```

Arguments:

nc_var The netCDF variable that describes this instance.

nc_dim The netCDF dimension that describes the dimensionality.

values Character vector of the label values.

Method print(): Prints a summary of the labels to the console.

Usage:

```
CFLabel$print(...)
```

Arguments:

`...` Arguments passed on to other functions. Of particular interest is `width =` to indicate a maximum width of attribute columns.

Returns: `self`, invisibly.

Method `subset()`**:** Retrieve a subset of the labels.

Usage:

```
CFLabel$subset(grp, rng)
```

Arguments:

`grp` The group to create the new label object in.

`rng` The range of indices to retrieve.

Returns: A `CFLabel` instance, or `NULL` if the `rng` values are invalid.

Method `write()`**:** Write the labels to a netCDF file, including its attributes.

Usage:

```
CFLabel$write(nc)
```

Arguments:

`nc` The handle of the netCDF file opened for writing or a group in the netCDF file. If `NULL`, write to the file or group where the labels were read from (the file must have been opened for writing). If not `NULL`, the handle to a netCDF file or a group therein.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFLabel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFObject` *CF base object*

Description

This class is a basic ancestor to all classes that represent CF objects, specifically data variables and axes. More useful classes use this class as ancestor.

Public fields

`NCvar` The NCVariable instance that this CF object represents.

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`id` (read-only) The identifier of the CF object.

`name` (read-only) The name of the CF object.

`fullname` (read-only) The fully-qualified name of the CF object.

`group` Retrieve the NCGroup that this object is located in.

`attributes` Set or retrieve a `data.frame` with the attributes of the CF object.

Methods

Public methods:

- `CFObject$new()`
- `CFObject$attribute()`
- `CFObject$print_attributes()`
- `CFObject$set_attribute()`
- `CFObject$append_attribute()`
- `CFObject$delete_attribute()`
- `CFObject$write_attributes()`
- `CFObject$add_coordinates()`
- `CFObject$clone()`

Method `new()`**:** Create a new CF object instance from a variable in a netCDF resource. This method is called upon opening a netCDF resource. It is rarely, if ever, useful to call this constructor directly from the console. Instead, use the methods from higher-level classes such as CFVariable.

Usage:

```
CFObject$new(nc_var)
```

Arguments:

`nc_var` The NCVariable instance upon which this CF object is based.

Returns: A `CFobject` instance.

Method `attribute()`**:** Retrieve attributes of any CF object.

Usage:

```
CFObject$attribute(att, field = "value")
```

Arguments:

`att` Vector of character strings of attributes to return.

`field` The field of the attribute to return values from. This must be "value" (default) or "type".

Returns: If the `field` argument is "type", a character string. If `field` is "value", a single value of the type of the attribute, or a vector when the attribute has multiple values. If no attribute is named with a value of argument `att` `NA` is returned.

Method `print_attributes()`**:** Print the attributes of the CF object to the console.

Usage:

```
CFObject$print_attributes(width = 50L)
```

Arguments:

`width` The maximum width of each column in the `data.frame` when printed to the console.

Method `set_attribute()`**:** Add an attribute. If an attribute `name` already exists, it will be overwritten.

Usage:

```
CFObject$set_attribute(name, type, value)
```

Arguments:

`name` The name of the attribute. The name must begin with a letter and be composed of letters, digits, and underscores, with a maximum length of 255 characters. UTF-8 characters are not supported in attribute names.

`type` The type of the attribute, as a string value of a netCDF data type.

`value` The value of the attribute. This can be of any supported type, including a vector or list of values. Matrices, arrays and like compound data structures should be stored as a data variable, not as an attribute and they are thus not allowed. In general, an attribute should be a character value, a numeric value, a logical value, or a short vector or list of any of these. Values passed in a list will be coerced to their common mode.

Returns: Self, invisibly.

Method `append_attribute()`**:** Append the text value of an attribute. If an attribute `name` already exists, the `value` will be appended to the existing value of the attribute. If the attribute `name` does not exist it will be created. The attribute must be of "NC_CHAR" or "NC_STRING" type; in the latter case having only a single string value.

Usage:

```
CFObject$append_attribute(name, value, sep = "; ", prepend = FALSE)
```

Arguments:

`name` The name of the attribute. The name must begin with a letter and be composed of letters, digits, and underscores, with a maximum length of 255 characters. UTF-8 characters are not supported in attribute names.

`value` The character value of the attribute to append. This must be a character string.

`sep` The separator to use. Default is `"; "`.

`prepend` Logical to flag if the supplied `value` should be placed before the existing value. Default is `FALSE`.

Returns: Self, invisibly.

Method `delete_attribute()`**:** Delete an attribute. If an attribute `name` is not present this method simply returns.

Usage:

```
CFObject$delete_attribute(name)
```

Arguments:

`name` The name of the attribute to delete.

Returns: Self, invisibly.

Method `write_attributes()`**:** Write the attributes of this object to a netCDF file.

Usage:

```
CFObject$write_attributes(nc, nm)
```

Arguments:

`nc` The handle to the netCDF file opened for writing.

`nm` The NC variable name or "NC_GLOBAL" to write the attributes to.

Returns: Self, invisibly.

Method `add_coordinates()`**:** Add names of axes to the "coordinates" attribute, avoiding duplicates and retaining previous values.

Usage:

```
CFObject$add_coordinates(crds)
```

Arguments:

`crds` Vector of axis names to add to the attribute.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFObject$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFResource` *NetCDF resource object*

Description

This class contains the connection details to a netCDF resource.

There is a single instance of this class for every netCDF resource, owned by the CFDataset instance. The instance is shared by other objects, specifically NCGroup and CFVariable instances, for access to the underlying resource for reading of data.

This class should never have to be accessed directly. All access is handled by higher-level methods.

Public fields

`error` Error message, or empty string.

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`handle` (read-only) The handle to the netCDF resource.

`uri` (read-only) The URI of the netCDF resource, either a local filename or the location of an online resource.

Methods

Public methods:

- `CFResource$new()`
- `CFResource$close()`
- `CFResource$group_handle()`
- `CFResource$clone()`

Method `new()`**:** Create a connection to a netCDF resource. This is called by `open_ncdf()` when opening a netCDF resource; you should never have to call this directly.

Usage:

```
CFResource$new(uri)
```

Arguments:

`uri` The URI to the netCDF resource.

Returns: An instance of this class.

Method `close()`**:** Closing an open netCDF resource. It should rarely be necessary to call this method directly.

Usage:

```
CFResource$close()
```

Method `group_handle()`**:** Every group in a netCDF file has its own handle, with the "root" group having the handle for the entire netCDF resource. The handle returned by this method is valid only for the named group.

Usage:

```
CFResource$group_handle(group_name)
```

Arguments:

`group_name` The absolute path to the group.

Returns: The handle to the group.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFResource$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

CFVariable *CF data variable*

Description

This class represents the basic structure of a CF data variable, the object that provides access to an array of data.

The CF data variable instance provides access to all the details that have been associated with the data variable, such as axis information, grid mapping parameters, etc. The actual data array can be accessed through the `data()` and `subset()` methods of this class.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFVariableBase` -> `CFVariable`

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`gridLongLat` The grid of longitude and latitude values of every grid cell when the main variable grid has a different coordinate system.

`crs_wkt2` (read-only) Retrieve the coordinate reference system description of the variable as a WKT2 string.

Methods

Public methods:

- `CFVariable$new()`
- `CFVariable$print()`
- `CFVariable$brief()`
- `CFVariable$shard()`
- `CFVariable$peek()`
- `CFVariable$data()`
- `CFVariable$subset()`
- `CFVariable$clone()`

Method `new()`**:** Create an instance of this class.

Usage:

```
CFVariable$new(nc_var, axes)
```

Arguments:

`nc_var` The netCDF variable that defines this CF variable.

`axes` List of CFAxis instances that describe the dimensions.

Returns: An instance of this class.

Method `print()`**:** Print a summary of the data variable to the console.

Usage:

```
CFVariable$print(...)
```

Arguments:

`...` Arguments passed on to other functions. Of particular interest is `width =` to indicate a maximum width of attribute columns.

Method `brief()`**:** Some details of the data variable.

Usage:

```
CFVariable$brief()
```

Returns: A 1-row `data.frame` with some details of the data variable.

Method `shard()`**:** The information returned by this method is very concise and most useful when combined with similar information from other variables.

Usage:

```
CFVariable$shard()
```

Returns: Character string with very basic variable information.

Method `peek()`**:** Retrieve interesting details of the data variable.

Usage:

```
CFVariable$peek(with_groups = TRUE)
```

Arguments:

`with_groups` Should group information be included? The save option is `TRUE` (default) when the netCDF resource has groups because names may be duplicated among objects in different groups.

Returns: A 1-row `data.frame` with details of the data variable.

Method `data()`**:** Retrieve all data of the variable.

Usage:

```
CFVariable$data()
```

Returns: A CFArray instance with all data from this variable.

Method `subset()`**:** This method extracts a subset of values from the array of the variable, with the range along each axis to extract expressed in coordinate values of the domain of each axis.

Usage:

```
CFVariable$subset(..., .aoi = NULL, rightmost.closed = FALSE)
```

Arguments:

`...` One or more arguments of the form `axis = range`. The "axis" part should be the name of an axis or its orientation `X`, `Y`, `Z` or `T`. The "range" part is a vector of values representing coordinates along the axis where to extract data. Axis designators and names are case-sensitive and can be specified in any order. If values for the range per axis fall outside of the extent of the axis, the range is clipped to the extent of the axis.

`.aoi` Optional, an area-of-interest instance of class `AOI` created with the `aoi()` function to indicate the horizontal area that should be extracted. The longitude and latitude coordinates must be included; the X and Y resolution will be calculated if not given. When provided, this argument will take precedence over the corresponding axis information for the X and Y axes in the `subset` argument.

`rightmost.closed` Single logical value to indicate if the upper boundary of range in each axis should be included.

Details: The range of values along each axis to be subset is expressed in coordinates of the domain of the axis. Any axes for which no selection is made in the `...` argument are extracted in whole. Coordinates can be specified in a variety of ways that are specific to the nature of the axis. For numeric axes it should (resolve to) be a vector of real values. A range (e.g. `100:200`), a vector (`c(23, 46, 3, 45, 17)`), a sequence (`seq(from = 78, to = 100, by = 2)`), all work. Note, however, that only a single range is generated from the vector so these examples resolve to `(100, 200)`, `(3, 46)`, and `(78, 100)`, respectively. For time axes a vector of character timestamps, `POSIXct` or `Date` values must be specified. As with numeric values, only the two extreme values in the vector will be used.
If the range of coordinate values for an axis in argument `...` extend the valid range of the axis in `x`, the extracted data will start at the beginning for smaller values and extend to the end for larger values. If the values envelope the valid range the entire axis will be extracted in the result. If the range of coordinate values for any axis are all either smaller or larger than the valid range of the axis then nothing is extracted and `NULL` is returned.
The extracted data has the same dimensional structure as the data in the variable, with degenerate dimensions dropped. The order of the axes in argument `...` does not reorder the axes in the result; use the CFArray$array() method for this.
As an example, to extract values of a variable for Australia for the year 2020, where the first axis in `x` is the longitude, the second axis is the latitude, both in degrees, and the third (and final) axis is time, the values are extracted by `x$subset(X = c(112, 154), Y = c(-9, -44), T = c("2020-01-01", "2021-01-01"))`. You could take the longitude-latitude values from `sf::st_bbox()` or `terra::ext()` if you have specific spatial geometries for whom you want to extract data. Note that this works equally well for projected coordinate reference systems - the key is that the specification in argument `...` uses the same domain of values as the respective axes in `x` use.

Auxiliary coordinate variables:
A special case exists for variables where the horizontal dimensions (X and Y) are not in longitude and latitude coordinates but in some other coordinate system. In this case the netCDF resource may have so-called *auxiliary coordinate variables* for longitude and latitude that are two grids with the same dimension as the horizontal axes of the data variable where each pixel gives the corresponding value for the longitude and latitude. If the variable has such *auxiliary coordinate variables* then they will be used automatically if, and only if, the axes are labeled in argument `...` as `X` and `Y`. The resolution of the grid that is produced by this method is automatically calculated. If you want to subset those axes then specify values in decimal degrees; if you want to extract the full extent, specify `NA` for both `X` and `Y`. **Note** that if you want to extract the data in the original grid, you should use the horizontal axis names in argument `....`

Returns: A CFArray instance, having an array with its axes and attributes of the variable, or `NULL` if one or more of the selectors in the `...` argument fall entirely outside of the range of the axis. Note that degenerate dimensions (having `length(.) == 1`) are dropped from the array but the corresponding axis is maintained in the result as a scalar axis.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFVariableBase` *Base ancestor of CFVariable and CFArray*

Description

This class is a basic ancestor to CFVariable and CFArray. It should not be instantiated directly, use the descendant classes instead.

This class provides access to common properties of data variables and the data they contain.

Super class

`ncdfCF::CFObject` -> `CFVariableBase`

Public fields

`axes` List of instances of classes descending from CFAxis that are the axes of the data object. If there are any scalar axes, they are listed after the axes that associate with the dimensions of the data. (In other words, axes `1..n` describe the `1..n` data dimensions, while any axes `n+1..m` are scalar axes.)

`crs` The coordinate reference system of this variable, as an instance of CFGridMapping. If this field is `NULL`, the horizontal component of the axes are in decimal degrees of longitude and latitude.

`cell_measure` The CFCellMeasure object of this variable, if defined.

Methods

Public methods:

- `CFVariableBase$new()`
- `CFVariableBase$time()`
- `CFVariableBase$summarise()`
- `CFVariableBase$profile()`
- `CFVariableBase$clone()`

Method `new()`**:** Create an instance of this class.

Usage:

```
CFVariableBase$new(var, axes, crs)
```

Arguments:

`var` The NC variable that describes this data object.

`axes` A `list` of CFAxis descendant instances that describe the axes of the data object.

`crs` The CFGridMapping instance of this data object, or `NULL` when no grid mapping is available.

Returns: An instance of this class.

Method `time()`**:** Return the time object from the axis representing time.

Usage:

```
CFVariableBase$time(want = "time")
```

Arguments:

`want` Character string with value "axis" or "time", indicating what is to be returned.

Returns: If `want = "axis"` the CFAxisTime axis; if `want = "time"` the `CFTime` instance of the axis, or `NULL` if the variable does not have a "time" axis.

Method `summarise()`**:** Summarise the temporal domain of the data, if present, to a lower resolution, using a user-supplied aggregation function.

Attributes are copied from the input data variable or data array. Note that after a summarisation the attributes may no longer be accurate. This method tries to sanitise attributes (such as removing `scale_factor` and `add_offset`, when present, as these will no longer be appropriate in most cases) but the onus is on the calling code (or yourself as interactive coder). Attributes like `standard_name` and `cell_methods` likely require an update in the output of this method, but the appropriate new values are not known to this method. Use `CFArray$set_attribute()` on the result of this method to set or update attributes as appropriate.

Usage:

```
CFVariableBase$summarise(name, fun, period, era = NULL, ...)
```

Arguments:

`name` Character vector with a name for each of the results that `fun` returns. So if `fun` has 2 return values, this should be a vector of length 2. Any missing values are assigned a default name of "result_#" (with '#' being replaced with an ordinal number).

`fun` A function or a symbol or character string naming a function that will be applied to each grouping of data. The function must return an atomic value (such as `sum()` or `mean()`), or a vector of atomic values (such as `range()`). Lists and other objects are not allowed and will throw an error that may be cryptic as there is no way that this method can assert that `fun` behaves properly so an error will pop up somewhere, most probably in unexpected ways. The function may also be user-defined so you could write a wrapper around a function like `lm()` to return values like the intercept or any coefficients from the object returned by calling that function.

`period` The period to summarise to. Must be one of either "day", "dekad", "month", "quarter", "season", "year". A "quarter" is the standard calendar quarter such as January-March, April-June, etc. A "season" is a meteorological season, such as December-February, March-May, etc. (any December data is from the year preceding the January data). The period must be of lower resolution than the resolution of the time axis.

`era` Optional, integer vector of years to summarise over by the specified `period`. The extreme values of the years will be used. This can also be a list of multiple such vectors. The elements in the list, if used, should have names as these will be used to label the results.

`...` Additional parameters passed on to `fun`.

Returns: A `CFData` object, or a list thereof with as many `CFData` objects as `fun` returns values.

Method `profile()`**:** This method extracts profiles of values from the array of the variable, with the location along each axis to extract expressed in coordinate values of each axis.

Usage:

```
CFVariableBase$profile(..., .names = NULL, .as_table = FALSE)
```

Arguments:

`...` One or more arguments of the form `axis = location`. The "axis" part should be the name of an axis or its orientation `X`, `Y`, `Z` or `T`. The "location" part is a vector of values representing coordinates along the axis where to profile. A profile will be generated for each of the elements of the "location" vectors in all arguments.

`.names` A character vector with names for the results. The names will be used for the `CFArray` instances, or as values for the "location" column of the `data.table` if argument `.as_table` is `TRUE`. If the vector is shorter than the longest vector of locations in the `...` argument, a name "location_#" will be used, with the # replaced by the ordinal number of the vector element.

`.as_table` Logical to flag if the results should be `CFArray` instances (`FALSE`, default) or a single `data.table` (`TRUE`). If `TRUE`, all `...` arguments must have the same number of elements, use the same axes and the `data.table` package must be installed.

Details: The coordinates along each axis to be sampled are expressed in values of the domain of the axis. Any axes which are not passed as arguments are extracted in whole to the result. If bounds are set on the axis, the coordinate whose bounds envelop the requested coordinate is selected. Otherwise, the coordinate along the axis closest to the supplied value will be used. If the value for a specified axis falls outside the valid range of that axis, `NULL` is returned.
A typical case is to extract the temporal profile as a 1D array for a given location. In this case, use arguments for the latitude and longitude on an X-Y-T data variable: `profile(lat = -24, lon = 3)`. Other profiling options are also possible, such as a 2D zonal atmospheric profile at a given longitude for an X-Y-Z data variable: `profile(lon = 34)`.
Multiple profiles can be extracted in one call by supplying vectors for the indicated axes: `profile(lat = c(-24, -23, -2), lon = c(5, 5, 6))`. The vectors need not have the same length, unless `.as_table = TRUE`. With unequal length vectors the result will be a `list` of CFArray instances with different dimensionality and/or different axes.

Auxiliary coordinate variables (CFVariable only):
A special case exists for variables where the horizontal dimensions (X and Y) are not in longitude and latitude coordinates but in some other coordinate system. In this case the netCDF resource may have so-called *auxiliary coordinate variables*. If the data variable has such *auxiliary coordinate variables* then they will be used automatically if, and only if, the axes are specified as `X` and `Y`. **Note** that if you want to profile the data in the original grid units, you should specify the horizontal axis names.

Returns: If `.as_table = FALSE`, a list of CFArray instances, each having one profile for each of the elements in the "location" vectors of argument `...` and named with the respective `.names` value. If `.as_table = TRUE`, a `data.table` with a row for each element along all profiles, with a ".variable" column using the values from the `.names` argument.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFVariableBase$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`CFVariableL3b` *CF data variable for the NASA L3b format*

Description

This class represents a CF data variable that provides access to data sets in NASA level-3 binned format, used extensively for satellite imagery.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFVariableBase` -> `ncdfCF::CFVariable` -> `CFVariableL3b`

Public fields

`variable` The name of the variable contained in this L3b data.

`index` The index data of the L3b structure.

Methods

Public methods:

- `CFVariableL3b$new()`
- `CFVariableL3b$as_matrix()`
- `CFVariableL3b$data()`
- `CFVariableL3b$subset()`
- `CFVariableL3b$clone()`

Method `new()`**:** Create an instance of this class.

Usage:

```
CFVariableL3b$new(grp, units)
```

Arguments:

`grp` The group that this CF variable lives in. Must be called "/level-3_binned_data".

`units` Vector of two character strings with the variable name and the physical units of the data variable in the netCDF resource.

Returns: An instance of this class.

Method `as_matrix()`**:** Read all the data from the file and turn the data into a matrix. If an `aoi` is specified, the data will be subset to that area.

This method returns a bare-bones matrix without any metadata or other identifying information. Use method `data()`, `subset()` or the `[` operator rather than this method to obtain a more informative result.

Usage:

```
CFVariableL3b$as_matrix(aoi = NULL)
```

Arguments:

`aoi` An instance of class `AOI`, optional, to select an area in latitude - longitude coordinates.

Returns: A matrix with the data of the variable in raw format.

Method `data()`**:** Retrieve all data of the L3b variable.

Usage:

```
CFVariableL3b$data()
```

Returns: A CFArray instance with all data from this L3b variable.

Method `subset()`**:** This method extracts a subset of values from the data of the variable, with the range along both axes expressed in decimal degrees.

Usage:

```
CFVariableL3b$subset(..., .aoi = NULL, rightmost.closed = FALSE)
```

Arguments:

`...` One or more arguments of the form `axis = range`. The "axis" part should be the name of axis `longitude` or `latitude` or its orientation `X` or `Y`. The "range" part is a vector of values representing coordinates along the axis where to extract data. Axis designators and names are case-sensitive and can be specified in any order. If values for the range of an axis fall outside of the extent of the axis, the range is clipped to the extent of the axis.

`.aoi` Optional, an area-of-interest instance of class `AOI` created with the `aoi()` function to indicate the horizontal area that should be extracted. The longitude and latitude coordinates must be included; the X and Y resolution will be calculated if not given. When provided, this argument will take precedence over the `...` argument.

`rightmost.closed` Single logical value to indicate if the upper boundary of range in each axis should be included.

Details: The range of values along both axes of latitude and longitude is expressed in decimal degrees. Any axes for which no information is provided in the `subset` argument are extracted in whole. Values can be specified in a variety of ways that should (resolve to) be a vector of real values. A range (e.g. `100:200`), a vector (`c(23, 46, 3, 45, 17)`, a sequence (`seq(from = 78, to = 100, by = 2)`, all work. Note, however, that only a single range is generated from the vector so these examples resolve to `(100, 200)`, `(3, 46)`, and `(78, 100)`, respectively.
If the range of values for an axis in argument `subset` extend the valid range of the axis in `x`, the extracted slab will start at the beginning for smaller values and extend to the end for larger values. If the values envelope the valid range the entire axis will be extracted in the result. If the range of `subset` values for any axis are all either smaller or larger than the valid range of the axis in `x` then nothing is extracted and `NULL` is returned.
The extracted data has the same dimensional structure as the data in the variable, with degenerate dimensions dropped. The order of the axes in argument `subset` does not reorder the axes in the result; use the CFArray$array() method for this.

Returns: A CFArray instance, having an array with axes and attributes of the variable, or `NULL` if one or more of the elements in the `...` argument falls entirely outside of the range of the axis. Note that degenerate dimensions (having `length(.) == 1`) are dropped from the array but the corresponding axis is maintained in the result as a scalar axis.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CFVariableL3b$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

https://oceancolor.gsfc.nasa.gov/resources/docs/technical/ocean_level-3_binned_data_products.pdf

---

`dim.AOI` *The dimensions of the grid of an AOI*

Description

This method returns the dimensions of the grid that would be created for the AOI.

Usage

```
## S3 method for class 'AOI'
dim(x)
```

Arguments

`x`	An instance of the `AOI` class.

Value

A vector of two values giving the longitude and latitude dimensions of the grid that would be created for the AOI.

Examples

```
a <- aoi(30, 40, 10, 30, 0.1)
dim(a)
```

dim.CFAxis *Axis length*

Description

This method returns the lengths of the axes of a variable or axis.

Usage

```
## S3 method for class 'CFAxis'
dim(x)
```

Arguments

x	The CFVariable or a descendant of CFAxis.

Value

Vector of axis lengths.

Examples

```
fn <- system.file("extdata", "ERA5land_Rwanda_20160101.nc", package = "ncdfCF")
ds <- open_ncdf(fn)
t2m <- ds[["t2m"]]
dim(t2m)
```

makeAxis *Create an axis*

Description

With this method you can create an axis to use with new CFArray instances. Depending on the orientation argument and the type of the values argument an instance of a class descending from CFAxis will be returned.

Usage

```
makeAxis(name, group, orientation, values, bounds = NULL)
```

Arguments

name	Name of the axis.
group	Group to place the axis in.
orientation	The orientation of the axis. Must be one of "X", "Y", "Z", or "T" for longitude, latitude, height or depth, and time axes, respectively. For any other axis, indicate an empty string ""
values	The coordinate values.
bounds	The bounds of the coordinate values, or NULL if not available.

Details

There are several restrictions on the combination of orientation and values arguments. Longitude and latitude axes (orientation of "X" or "Y") must have numeric values. For a time axis (orientation of "T") the values argument must be an instance of CFTime or CFClimatology.

Value

An instance of a class descending from CFAxis.

See Also

makeLongitudeAxis(), makeLatitudeAxis(), makeTimeAxis(), makeDiscreteAxis()

---

makeDiscreteAxis *Create a discrete axis*

Description

With this method you can create a discrete axis to use with new CFArray instances.

Usage

```
makeDiscreteAxis(name, group, length)
```

Arguments

name	Name of the axis.
group	Group to place the axis in.
length	The length of the axis.

Value

A CFAxisDiscrete instance. The values will be a sequence of size length.

makeGroup *Create a group in memory to hold CF objects*

Description

With this function a group is created in memory, i.e. not associated with a netCDF resource on file. This can be used to prepare new CF objects before writing them to file. Extracting data from a `CFVariable` into a CFArray instance will also create a virtual group.

Usage

```
makeGroup(id = -1L, name = "/", fullname = "/", parent = NULL)
```

Arguments

`id`	The id of the group, default -1L.
`name`	The name of the group, default "/".
`fullname`	The full path and name of the group, default "/".
`parent`	Optionally, a parent group to which the new group will be added as a child.

Value

A `NCGroup` instance.

makeLatitudeAxis *Create a latitude axis*

Description

With this method you can create a latitude axis to use with new CFArray instances.

Usage

```
makeLatitudeAxis(name, group, values, bounds)
```

Arguments

`name`	Name of the axis.
`group`	Group to place the axis in.
`values`	The coordinate values.
`bounds`	The bounds of the coordinate values, or `NULL` if not available.

Value

A CFAxisLatitude instance.

makeLongitudeAxis *Create a longitude axis*

Description

With this method you can create a longitude axis to use with new CFArray instances.

Usage

```
makeLongitudeAxis(name, group, values, bounds = NULL)
```

Arguments

name	Name of the axis.
group	Group to place the axis in.
values	The coordinate values.
bounds	The bounds of the coordinate values, or NULL if not available.

Value

A CFAxisLongitude instance.

makeTimeAxis *Create a time axis*

Description

With this method you can create a time axis to use with new CFArray instances.

Usage

```
makeTimeAxis(name, group, values)
```

Arguments

name	Name of the axis.
group	Group to place the axis in.
values	A CFTime instance with time values and optionally bounds set.

Value

A CFAxisTime instance.

`names.CFDataset` *Names or dimension values of an CF object*

Description

Retrieve the variable or dimension names of an `ncdfCF` object. The `names()` function gives the names of the variables in the data set, prepended with the path to the group if the resource uses groups. The return value of the `dimnames()` function differs depending on the type of object:

- `CFDataset`, `CFVariable`: The dimnames are returned as a vector of the names of the axes of the data set or variable, prepended with the path to the group if the resource uses groups. Note that this differs markedly from the `base::dimnames()` functionality.
- `CFAxisNumeric`, `CFAxisLongitude`, `CFAxisLatitude`, `CFAxisVertical`: The values of the elements along the axis as a numeric vector.
- `CFAxisTime`: The values of the elements along the axis as a character vector containing timestamps in ISO8601 format. This could be dates or date-times if time information is available in the axis.
- `CFAxisCharacter`: The values of the elements along the axis as a character vector.
- `CFAxisDiscrete`: The index values of the axis, from 1 to the length of the axis.

Usage

```
## S3 method for class 'CFDataset'
names(x)

groups(x)

## S3 method for class 'CFDataset'
groups(x)
```

Arguments

`x`	An `CFObject` whose axis names to retrieve. This could be `CFDataset`, `CFVariable`, or a class descending from `CFAxis`.

Value

A vector as described in the Description section.

Examples

```
fn <- system.file("extdata",
  "pr_day_EC-Earth3-CC_ssp245_r1i1p1f1_gr_20230101-20231231_vncdfCF.nc",
  package = "ncdfCF")
ds <- open_ncdf(fn)

# CFDataset
```

```
dimnames(ds)

# CFVariable
pr <- ds[["pr"]]
dimnames(pr)

# CFAxisNumeric
lon <- ds[["lon"]]
dimnames(lon)

# CFAxisTime
t <- ds[["time"]]
dimnames(t)
```

`NCDimension` *NetCDF dimension object*

Description

This class represents an netCDF dimensions. It contains the information on a dimension that is stored in an netCDF file.

This class is not very useful for interactive use. Use the CFAxis descendent classes instead.

Super class

`ncdfCF::NCObject` -> `NCDimension`

Public fields

`length` The length of the dimension. If field `unlim` = `TRUE`, this field indicates the length of the data in this dimension written to file.

`unlim` Logical flag to indicate if the dimension is unlimited, i.e. that additional data may be written to file incrementing in this dimension.

Methods

Public methods:

- `NCDimension$new()`
- `NCDimension$print()`
- `NCDimension$shard()`
- `NCDimension$write()`
- `NCDimension$clone()`

Method `new()`**:** Create a new netCDF dimension. This class should not be instantiated directly, create CF objects instead. This class is instantiated when opening a netCDF resource.

Usage:

```
NCDimension$new(id, name, length, unlim)
```

Arguments:

`id` Numeric identifier of the netCDF dimension.

`name` Character string with the name of the netCDF dimension.

`length` Length of the dimension.

`unlim` Is the dimension unlimited?

Returns: A `NCDimension` instance.

Method `print()`**:** Summary of the NC dimension printed to the console.

Usage:

```
NCDimension$print(...)
```

Arguments:

`...` Passed on to other methods.

Method `shard()`**:** Very concise information on the dimension. The information returned by this function is very concise and most useful when combined with similar information from other dimensions.

Usage:

```
NCDimension$shard()
```

Returns: Character string with very basic dimension information.

Method `write()`**:** Write the dimension to a netCDF file.

Usage:

```
NCDimension$write(h)
```

Arguments:

`h` The handle to the netCDF file to write.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NCDimension$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`NCGroup` *NetCDF group*

Description

This class represents a netCDF group, the object that holds elements like dimensions and variables of a netCDF file. This class also holds references to any CF objects based on the netCDF elements held by the group.

Direct access to groups is usually not necessary. The principal objects held by the group, CF data variables and axes, are accessible via other means. Only for access to the group attributes is a reference to a group required.

Super class

ncdfCF::NCObject -> NCGroup

Public fields

`resource` Access to the underlying netCDF resource. This can be `NULL` for instances created in memory.

`fullname` The fully qualified absolute path of the group.

`parent` Parent group of this group, the owning `CFDataset` for the root group.

`subgroups` List of child `NCGroup` instances of this group.

`NCvars` List of netCDF variables that are located in this group.

`NCdims` List of netCDF dimensions that are located in this group.

`NCudts` List of netCDF user-defined types that are located in this group.

`CFvars` List of CF data variables in this group. There must be a corresponding item in `NCvars` for each item in this list.

`CFaxes` List of axes of CF data variables in this group. There must be a corresponding item in `NCvars` for each item in this list. Note that the CF data variable(s) that an axis is associated with may be located in a different group. Also, objects that further describe the basic axis definition, such as its bounds, labels, ancillary data, may be located in a different group; all such elements can be accessed directly from the CFAxis instances that this list holds.

`CFaux` List of auxiliary coordinates located in this group. These could be CFLabel instances or an axis.

`CFlonglat` List of CFAuxiliaryLongLat that hold longitude and latitude values for every grid point in the data variable that references them.

`CFmeasures` List of cell measures variables in this group.

`CFcrs` List of grid mappings located in this group.

Active bindings

`friendlyClassName` (read-only) A nice description of the class.

`handle` (read-only) Get the handle to the netCDF resource for the group

`root` (read-only) Retrieve the root group.

`data_set` (read-only) Retrieve the CFDataset that the group belongs to.

Methods

Public methods:

- `NCGroup$new()`
- `NCGroup$print()`
- `NCGroup$hierarchy()`
- `NCGroup$find_by_name()`
- `NCGroup$find_dim_by_id()`
- `NCGroup$has_name()`

- NCGroup$unused()
- NCGroup$addAuxiliaryLongLat()
- NCGroup$addCellMeasure()
- NCGroup$fullnames()
- NCGroup$dimensions()
- NCGroup$variables()
- NCGroup$axes()
- NCGroup$grid_mappings()
- NCGroup$clone()

Method `new()`**:** Create a new instance of this class.

Usage:

```
NCGroup$new(id, name, fullname, parent, resource)
```

Arguments:

`id` The identifier of the group.

`name` The name of the group.

`fullname` The fully qualified name of the group.

`parent` The parent group of this group. `NULL` for the root group.

`resource` Reference to the CFResource instance that provides access to the netCDF resource. For in-memory groups this can be `NULL`.

Method `print()`**:** Summary of the group printed to the console.

Usage:

```
NCGroup$print(stand_alone = TRUE, ...)
```

Arguments:

`stand_alone` Logical to indicate if the group should be printed as an object separate from other objects (`TRUE`, default), or print as part of an enclosing object (`FALSE`).

`...` Passed on to other methods.

Method `hierarchy()`**:** Prints the hierarchy of the group and its subgroups to the console, with a summary of contained objects. Usually called from the root group to display the full group hierarchy.

Usage:

```
NCGroup$hierarchy(idx = 1L, total = 1L)
```

Arguments:

`idx, total` Arguments to control indentation. Should both be 1 (the default) when called interactively. The values will be updated during recursion when there are groups below the current group.

Method `find_by_name()`**:** Find an object by its name. Given the name of an object, possibly preceded by an absolute or relative group path, return the object to the caller. Typically, this method is called programmatically; similar interactive use is provided through the `[[.CFDataset` operator.

Usage:

```
NCGroup$find_by_name(name, scope = "CF")
```

Arguments:

`name` The name of an object, with an optional absolute or relative group path from the calling group. The object must either an CF construct (data variable, axis, auxiliary axis, label, or grid mapping) or an NC group, dimension or variable.

`scope` Either "CF" (default) for a CF construct, or "NC" for a netCDF group, dimension or variable.

Returns: The object with the provided name in the requested scope. If the object is not found, returns `NULL`.

Method `find_dim_by_id()`**:** Find an NC dimension object by its id. Given the id of a dimension, return the NCDimension object to the caller. The dimension has to be found in the current group or any of its parents.

Usage:

```
NCGroup$find_dim_by_id(id)
```

Arguments:

`id` The id of the dimension.

Returns: The NCDimension object with an identifier equal to the `id` argument. If the object is not found, returns `NULL`.

Method `has_name()`**:** Has a given name been defined in this group already?

Usage:

```
NCGroup$has_name(name, scope = "both")
```

Arguments:

`name` Character string. The name will be searched for, regardless of case.

`scope` Either "CF" for a CF construct, "NC" for a netCDF variable, or "both" (default) to test both scopes.

Returns: `TRUE` if `name` is present in the group, `FALSE` otherwise.

Method `unused()`**:** Find NC variables that are not referenced by CF objects. For debugging purposes only.

Usage:

```
NCGroup$unused()
```

Returns: List of NCVariable.

Method `addAuxiliaryLongLat()`**:** Add an auxiliary long-lat variable to the group. This method creates a CFAuxiliaryLongLat from the arguments and adds it to the group `CFlonglat` list, but only if the combination of `lon`, `lat` isn't already present.

Usage:

```
NCGroup$addAuxiliaryLongLat(lon, lat, bndsLong, bndsLat)
```

Arguments:

`lon, lat` Instances of NCVariable having a two-dimensional grid of longitude and latitude values, respectively.

`bndsLong`, `bndsLat` Instances of CFBounds with the 2D bounds of the longitude and latitude grid values, respectively, or `NULL` when not set.

Returns: `self` invisibly.

Method `addCellMeasure()`**:** Add a cell measure variable to the group.

Usage:

```
NCGroup$addCellMeasure(cm)
```

Arguments:

`cm` Instance of CFCellMeasure.

Returns: `self` invisibly.

Method `fullnames()`**:** This method lists the fully qualified name of this group, optionally including names in subgroups.

Usage:

```
NCGroup$fullnames(recursive = TRUE)
```

Arguments:

`recursive` Should subgroups be scanned for names too (default is `TRUE`)?

Returns: A character vector with group names.

Method `dimensions()`**:** List all the dimensions that are visible from this group including those that are defined in parent groups (by names not defined by any of their child groups in direct lineage to the current group).

Usage:

```
NCGroup$dimensions()
```

Returns: A vector of NCDimension objects.

Method `variables()`**:** This method lists the CF data variables located in this group, optionally including data variables in subgroups.

Usage:

```
NCGroup$variables(recursive = TRUE)
```

Arguments:

`recursive` Should subgroups be scanned for CF data variables too (default is `TRUE`)?

Returns: A list of CFVariable.

Method `axes()`**:** This method lists the axes located in this group, optionally including axes in subgroups.

Usage:

```
NCGroup$axes(recursive = TRUE)
```

Arguments:

`recursive` Should subgroups be scanned for axes too (default is `TRUE`)?

Returns: A list of CFAxis descendants.

Method `grid_mappings()`**:** This method lists the grid mappings located in this group, optionally including grid mappings in subgroups.

Usage:

```
NCGroup$grid_mappings(recursive = TRUE)
```

Arguments:

`recursive` Should subgroups be scanned for grid mappings too (default is `TRUE`)?

Returns: A list of CFGridMapping instances.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NCGroup$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`NCObject` *NetCDF base object*

Description

This class is a basic ancestor to all classes that represent netCDF objects, specifically groups, dimensions, variables and the user-defined types in a netCDF file. More useful classes use this class as ancestor.

The fields in this class are common among all netCDF objects. In addition, this class manages the attributes for its descendent classes.

Public fields

`id` Numeric identifier of the netCDF object.

`name` The name of the netCDF object.

`attributes` `data.frame` with the attributes of the netCDF object.

Methods

Public methods:

- `NCObject$new()`
- `NCObject$print_attributes()`
- `NCObject$attribute()`
- `NCObject$set_attribute()`
- `NCObject$append_attribute()`
- `NCObject$delete_attribute()`
- `NCObject$write_attributes()`
- `NCObject$add_coordinates()`

- NCObject$clone()

Method `new()`**:** Create a new netCDF object. This class should not be instantiated directly, create descendant objects instead.

Usage:

```
NCObject$new(id, name)
```

Arguments:

`id` Numeric identifier of the netCDF object.

`name` Character string with the name of the netCDF object.

Method `print_attributes()`**:** This function prints the attributes of the netCDF object to the console. Through object linkages, this also applies to the CF data variables and axes, which each link to a netCDF object.

Usage:

```
NCObject$print_attributes(width = 50L)
```

Arguments:

`width` The maximum width of each column in the `data.frame` when printed to the console.

Method `attribute()`**:** This method returns an attribute of a netCDF object.

Usage:

```
NCObject$attribute(att, field = "value")
```

Arguments:

`att` Attribute name whose value to return.

`field` The field of the attribute to return values from. This must be "value" (default) or "type".

Returns: If the `field` argument is "type", a character string. If `field` is "value", a single value of the type of the attribute, or a vector when the attribute has multiple values. If no attribute is named with a value of argument `att` `NA` is returned.

Method `set_attribute()`**:** Add an attribute. If an attribute `name` already exists, it will be overwritten.

Usage:

```
NCObject$set_attribute(name, type, value)
```

Arguments:

`name` The name of the attribute. The name must begin with a letter and be composed of letters, digits, and underscores, with a maximum length of 255 characters. UTF-8 characters are not supported in attribute names.

`type` The type of the attribute, as a string value of a netCDF data type or a user-defined type.

`value` The value of the attribute. This can be of any supported type, including a vector or list of values. Matrices, arrays and like compound data structures should be stored as a data variable, not as an attribute and they are thus not allowed. In general, an attribute should be a character value, a numeric value, a logical value, or a short vector or list of any of these. Values passed in a list will be coerced to their common mode.

Returns: Self, invisibly.

Method `append_attribute()`**:** Append the text value of an attribute. If an attribute `name` already exists, the `value` will be appended to the existing value of the attribute. If the attribute `name` does not exist it will be created. The attribute must be of "NC_CHAR" or "NC_STRING" type; in the latter case having only a single string value.

Usage:

```
NCObject$append_attribute(name, value, sep = "; ", prepend = FALSE)
```

Arguments:

`name` The name of the attribute. The name must begin with a letter and be composed of letters, digits, and underscores, with a maximum length of 255 characters. UTF-8 characters are not supported in attribute names.

`value` The character value of the attribute to append. This must be a character string.

`sep` The separator to use. Default is `"; "`.

`prepend` Logical to flag if the supplied `value` should be placed before the existing value. Default is `FALSE`.

Returns: Self, invisibly.

Method `delete_attribute()`**:** Delete attributes. If an attribute `name` is not present this method simply returns.

Usage:

```
NCObject$delete_attribute(name)
```

Arguments:

`name` Vector of names of the attributes to delete.

Returns: Self, invisibly.

Method `write_attributes()`**:** Write the attributes of this object to a netCDF file.

Usage:

```
NCObject$write_attributes(nc, nm)
```

Arguments:

`nc` The handle to the netCDF file opened for writing.

`nm` The NC variable name or "NC_GLOBAL" to write the attributes to.

Returns: Self, invisibly.

Method `add_coordinates()`**:** Add names of axes to the "coordinates" attribute, avoiding duplicates and retaining previous values.

Usage:

```
NCObject$add_coordinates(crds)
```

Arguments:

`crds` Vector of axis names to add to the attribute.

Returns: Self, invisibly.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NCObject$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

NCUDT *NetCDF user-defined type*

Description

This class represents user-defined types in a netCDF file. Interpretation of the UDT typically requires knowledge of the data set or application.

Super class

`ncdfCF::NCObject` -> NCUDT

Public fields

`clss` The class of the UDT, one of "builtin", "compound", "enum", "opaque", or "vlen".

`size` Size in bytes of a single item of the type (or a single element of a "vlen").

`basetype` Name of the netCDF base type of each element ("enum" and "vlen" only).

`value` Named vector with numeric values of all members ("enum" only).

`offset` Named vector with the offset of each field in bytes from the beginning of the "compound" type.

`subtype` Named vector with the netCDF base type name of each field of a "compound" type.

`dimsizes` Named list with array dimensions of each field of a "compound" type. A `NULL` length indicates a scalar.

Methods

Public methods:

- `NCUDT$new()`
- `NCUDT$clone()`

Method `new()`**:** Create a new netCDF user-defined type. This class represents a user-defined type. It is instantiated when opening a netCDF resource.

Usage:

```
NCUDT$new(id, name, clss, size, basetype, value, offset, subtype, dimsizes)
```

Arguments:

`id` Numeric identifier of the user-defined type.

`name` Character string with the name of the user-defined type.

`clss` The class of the UDT, one of "builtin", "compound", "enum", "opaque", or "vlen".

`size` Size in bytes of a single item of the type (or a single element of a "vlen").

`basetype` Name of the netCDF base type of each element ("enum" and "vlen" only).

`value` Named vector with numeric values of all members ("enum" only).

`offset` Named vector with the offset of each field in bytes from the beginning of the "compound" type.

`subtype` Named vector with the netCDF base type name of each field of a "compound" type.

`dimsizes` Named list with array dimensions of each field of a "compound" type. A `NULL` length indicates a scalar.

Returns: An instance of this class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NCUDT$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`NCVariable` *NetCDF variable*

Description

This class represents a netCDF variable, the object that holds the properties and data of elements like dimensions and variables of a netCDF file.

Direct access to netCDF variables is usually not necessary. NetCDF variables are linked from CF data variables and axes and all relevant properties are thus made accessible.

Super class

`ncdfCF::NCObject` -> `NCVariable`

Public fields

`group` NetCDF group where this variable is located.

`vtype` The netCDF data type of this variable. This could be the packed type. Don't check this field but use the appropriate method in the class of the object whose data type you are looking for.

`ndims` Number of dimensions that this variable uses.

`dimids` Vector of dimension identifiers that this variable uses. These are the so-called "NUG coordinate variables".

`netcdf4` Additional properties for a `netcdf4` resource.

Active bindings

`CF` List of CF objects that use this netCDF variable.

`fullname` (read-only) Name of the NC variable including the group path from the root group.

Methods

Public methods:

- NCVariable$new()
- NCVariable$print()
- NCVariable$shard()
- NCVariable$clone()

Method `new()`**:** Create a new netCDF variable. This class should not be instantiated directly, they are created automatically when opening a netCDF resource.

Usage:

```
NCVariable$new(id, name, group, vtype, ndims, dimids)
```

Arguments:

`id` Numeric identifier of the netCDF object.

`name` Character string with the name of the netCDF object.

`group` The NCGroup this variable is located in.

`vtype` The netCDF data type of the variable.

`ndims` The number of dimensions this variable uses.

`dimids` The identifiers of the dimensions this variable uses.

Returns: An instance of this class.

Method `print()`**:** Summary of the NC variable printed to the console.

Usage:

```
NCVariable$print(...)
```

Arguments:

`...` Passed on to other methods.

Method `shard()`**:** Very concise information on the variable. The information returned by this function is very concise and most useful when combined with similar information from other variables.

Usage:

```
NCVariable$shard()
```

Returns: Character string with very basic variable information.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
NCVariable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

open_ncdf *Open a netCDF resource*

Description

This function will read the metadata of a netCDF resource and interpret the netCDF dimensions, variables and attributes to generate the corresponding CF objects. The data for the CF variables is not read, please see CFVariable for methods to read the variable data.

Usage

```
open_ncdf(resource, keep_open = FALSE)
```

Arguments

resource	The name of the netCDF resource to open, either a local file name or a remote URI.
keep_open	Logical flag to indicate if the netCDF resource has to remain open after reading the metadata. This should be enabled typically only for programmatic access or when a remote resource has an expensive access protocol (i.e. 2FA). The resource has to be explicitly closed with `close()` after use. Note that when a data set is opened with `keep_open = TRUE` the resource may still be closed by the operating system or the remote server.

Value

An `CFDataset` instance, or an error if the resource was not found or errored upon reading.

Examples

```
fn <- system.file("extdata",
  "pr_day_EC-Earth3-CC_ssp245_r1i1p1f1_gr_20230101-20231231_vncdfCF.nc",
  package = "ncdfCF")
(ds <- open_ncdf(fn))
```

peek_ncdf *Examine a netCDF resource*

Description

This function will read a netCDF resource and return a list of identifying information, including data variables, axes and global attributes. Upon returning the netCDF resource is closed.

Usage

```
peek_ncdf(resource)
```

Arguments

resource	The name of the netCDF resource to open, either a local file name or a remote URI.

Details

If you find that you need other information to be included in the result, open an issue: https://github.com/pvanlaake/ncdfCF/iss

Value

A list with elements "variables", "axes" and global "attributes", each a data.frame.

Examples

```
fn <- system.file("extdata",
  "pr_day_EC-Earth3-CC_ssp245_r1i1p1f1_gr_20230101-20231231_vncdfCF.nc",
  package = "ncdfCF")
peek_ncdf(fn)
```

---

str.CFAxis *Compact display of an axis.*

Description

Compact display of an axis.

Usage

```
## S3 method for class 'CFAxis'
str(object, ...)
```

Arguments

object	A CFAxis instance or any descendant.
...	Ignored.

str.CFDataset *Compact display of a CFDataset*

Description

Compact display of a CFDataset

Usage

```
## S3 method for class 'CFDataset'
str(object, ...)
```

Arguments

object	A CFDataset instance.
...	Ignored.

[.CFVariable *Extract data for a variable*

Description

Extract data from a CFVariable instance, optionally sub-setting the axes to load only data of interest.

Usage

```
## S3 method for class 'CFVariable'
x[i, j, ..., drop = FALSE]
```

Arguments

x	An CFVariable instance to extract the data of.
i, j, ...	Expressions, one for each axis of x, that select a number of elements along each axis. If any expressions are missing, the entire axis is extracted. The values for the arguments may be an integer vector or a function that returns an integer vector. The range of the values in the vector will be used. See examples, below.
drop	Logical, ignored. Axes are never dropped. Any degenerate dimensions of the array are returned as such, with dimnames and appropriate attributes set.

Details

If all the data of the variable in `x` is to be extracted, simply use `[]` (unlike with regular arrays, this is required, otherwise the details of the variable are printed on the console).

The indices into the axes to be subset can be specified in a variety of ways; in practice it should (resolve to) be a vector of integers. A range (e.g. `100:200`), an explicit vector (`c(23, 46, 3, 45, 17)`), a sequence (`seq(from = 78, to = 100, by = 2)`), all work. Note, however, that only a single range is generated from the vector so these examples resolve to `100:200`, `3:46`, and `78:100`, respectively. It is also possible to use a custom function as an argument.

This method works with "bare" indices into the axes of the array. If you want to use domain values of the axes (e.g. longitude values or timestamps) to extract part of the variable array, use the `CFVariable$subset()` method.

Scalar axes should not be included in the indexing as they do not represent a dimension into the data array.

Value

An array with dimnames and other attributes set.

Examples

```
fn <- system.file("extdata",
  "pr_day_EC-Earth3-CC_ssp245_r1i1p1f1_gr_20230101-20231231_vncdfCF.nc",
  package = "ncdfCF")
ds <- open_ncdf(fn)
pr <- ds[["pr"]]

# How are the dimensions organized?
dimnames(pr)

# Precipitation data for March for a single location
x <- pr[5, 12, 61:91]
str(x)

# Summer precipitation over the full spatial extent
summer <- pr[, , 173:263]
str(summer)
```

`[.CFVariableL3b` *Extract data for a variable*

Description

Extract data from a `CFVariableL3b` instance, optionally sub-setting the axes to load only data of interest.

Usage

```
## S3 method for class 'CFVariableL3b'
x[i, j, ..., drop = FALSE]
```

Arguments

x	An CFVariableL3b instance to extract the data of.
i, j, ...	Expressions, one for each of the two axes of x, that select a number of elements along each axis. i is for the longitude axis, j for the latitude axis, ... (additional named arguments) is invalid as there are only two axes to subset from. If either expression is missing, the entire axis is extracted. The values for the arguments may be an integer vector or a function that returns an integer vector. The range of the values in the vector will be used. See examples, below.
drop	Logical, ignored. Axes are never dropped. Any degenerate dimensions of the array are returned as such, with dimnames and appropriate attributes set.

Details

If all the data of the variable in x is to be extracted, simply use [] (unlike with regular arrays, this is required, otherwise the details of the variable are printed on the console).

The indices into the axes to be subset can be specified in a variety of ways; in practice it should (resolve to) be a vector of integers. A range (e.g. 100:200), an explicit vector (c(23, 46, 3, 45, 17), a sequence (seq(from = 78, to = 100, by = 2), all work. Note, however, that only a single range is generated from the vector so these examples resolve to 100:200, 3:46, and 78:100, respectively. It is also possible to use a custom function as an argument.

This method works with "bare" indices into the axes of the array. If you want to use domain values of the axes (e.g. longitude values or timestamps) to extract part of the variable array, use the CFVariableL3b$subset() method.

Scalar axes should not be included in the indexing as they do not represent a dimension into the data array.

Value

An array with dimnames and other attributes set.

Examples

```
fn <- system.file("extdata",
  "pr_day_EC-Earth3-CC_ssp245_r1i1p1f1_gr_20230101-20231231_vncdfCF.nc",
  package = "ncdfCF")
ds <- open_ncdf(fn)
pr <- ds[["pr"]]

# How are the dimensions organized?
dimnames(pr)

# Precipitation data for March for a single location
x <- pr[5, 12, 61:91]
```

```
str(x)

# Summer precipitation over the full spatial extent
summer <- pr[, , 173:263]
str(summer)
```

`[[.CFDataset` *Get a variable or axis object from a data set*

Description

This method can be used to retrieve a variable or axis from the data set by name.

Usage

```
## S3 method for class 'CFDataset'
x[[i]]
```

Arguments

`x`	An `CFDataset` to extract a variable or axis from.
`i`	The name of a variable or axis in `x`. If data set `x` has groups, `i` should be an absolute path to the object to retrieve.

Details

If the data set has groups, the name `i` of the variable or axis should be fully qualified with the path to the group where the object is located. This fully qualified name can be retrieved with the `names()` and `dimnames()` functions, respectively.

Value

An instance of `CFVariable` or an `CFAxis` descendant class, or `NULL` if the name is not found.

Examples

```
fn <- system.file("extdata", "ERA5land_Rwanda_20160101.nc", package = "ncdfCF")
ds <- open_ncdf(fn)
v1 <- names(ds)[1]
var <- ds[[v1]]
var
```

Index