Package: ncdfCF (via r-universe)

June 7, 2026

Type Package

Title Easy Access to NetCDF Files with CF Metadata Conventions

Version 0.8.2.9000

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

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports abind, CFtime, methods, R6, RNetCDF, stringr

Collate 'AOI.R' 'AOImethod.R' 'CFAuxiliaryLongLat.R' 'CFAxis.R' 'CFAxisCharacter.R' 'CFAxisDiscrete.R' 'CFAxisLatitude.R' 'CFAxisLongitude.R' 'CFAxisNumeric.R' 'CFAxisTime.R' 'CFAxisVertical.R' 'CFBounds.R' 'CFCellMeasure.R' 'CFData.R' 'CFDataset.R' 'CFGridMapping.R' 'CFGroup.R' 'CFLabel.R' 'NCObject.R' 'CFObject.R' 'CFStandardNames.R' 'CFVariable.R' 'CFVariableL3b.R' 'CFVerticalParametricTerm.R' 'NCDimension.R' 'NCGroup.R' 'NCResource.R' 'NCVariable.R' 'makeCFObjects.R' 'ncdfCF-package.R' 'ops.R' 'plot.R' 'readCF.R' 'utils.R' 'wkt2.R' 'zzz.R'

Suggests data.table, ggplot2, knitr, rmarkdown, stars, terra, testthat (>= 3.0.0), units

VignetteBuilder knitr

Depends R (>= 4.0)

URL https://github.com/R-CF/ncdfCF, https://r-cf.github.io/ncdfCF/

BugReports https://github.com/R-CF/ncdfCF/issues

Config/testthat/edition 3

Config/Needs/website rmarkdown

Config/pak/sysreqs libicu-dev libnetcdf-dev libudunits2-dev

Repository https://r-cf.r-universe.dev

Date/Publication 2026-06-07 16:01:04 UTC

RemoteUrl https://github.com/r-cf/ncdfcf

RemoteRef HEAD

RemoteSha 74bfb30d89f5815eaa46f66577f8ec68e95c1a29

Contents

`[.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 CF 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 <- ds$var_names[1]
var <- ds[[v1]]
var
```

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 recommended values lie within the range `[0.01 ... 10]` - if the values fall outside of this range an warning will be issued. 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))
```

as_CF *Create a* CFDataset *or* CFVariable *instance from an R object*

Description

With this function you can convert an R object into a CFDataset or CFVariable, depending on the characteristics of the argument obj. The object to convert can be an array, matrix or vector of type logical, integer, numeric or character, or a terra::SpatRaster.

Usage

```
as_CF(name, obj)

## Default S3 method:
as_CF(name, obj)

## S3 method for class 'SpatRaster'
as_CF(name, obj)
```

Arguments

name	The name of the CFDataset or CFVariable to create.
obj	The object to convert. This can be an array, matrix or vector of type logical, integer, numeric or character, or a terra::SpatRaster.

Details

Dimnames on the R object will be converted to instances of a CFAxis descendant class, depending on their values. If the dimnames along a dimension of the R object can be converted to numeric, then it will be an instance of CFAxisNumeric. If the dimnames are character, a first attempt is made to create a CFAxisTime (i.e. the dimnames have to represent timestamps), failing that a CFAxisCharacter will be created. If no dimnames are set, an instance of CFAxisDiscrete is generated.

The axes of the CFVariable instance(s) are oriented as in the object. Note that this is different from standard practice in the netCDF community and the portability of saved data sets is thus limited. You can improve this situation by setting the orientation of the axes and by adding attributes.

After creation of the CFDataset or CFVariable, it is recommended to set other properties, such as attributes or a coordinate reference system.

Value

An instance of class CFDataset or CFVariable.

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 of this class manually therefore has no practical purpose.

Active bindings

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

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

`grid_names` (read-only) Read the names of the longitude and latitude grid as a vector of length 2.

`dimids` (read-only) Retrieve the dimension ids used by the longitude and latitude grids.

`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.

`lon_bounds` (read-only) Retrieve the boundary values of the longitude grid.

`lat_bounds` (read-only) Retrieve the boundary values of 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 minimum and maximum longitude and minimum and maximum latitude.

`dim` (read-only) The dimensions 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$attach_to_group()`
- `CFAuxiliaryLongLat$detach()`
- `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 = NULL, boundsLat = NULL)
```

Arguments:

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

`boundsLong, boundsLat` The CFBounds instances of the grid cells for the longitude and latitude, respectively, if set. Defaults to `NULL`.

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()
```

Method `attach_to_group()`**:** Attach the auxiliary long-lat grids and any bounds to a group. If there is another object with the same name in this group an error is thrown.

Usage:

```
CFAuxiliaryLongLat$attach_to_group(grp, locations = list())
```

Arguments:

`grp` An instance of CFGroup.

`locations` Optional. A `list` whose named elements correspond to the names of objects associated with these auxiliary grids. Each list element has a single character string indicating the group in the hierarchy where the object should be stored. As an example, if the variable has axes "lon" and "lat" and they should be stored in the parent group of `grp`, then specify `locations = list(lon = "..", lat = "..")`. Locations can use absolute paths or relative paths from group `grp`. The auxiliary grids and bounds that are not in the list will be stored in group `grp`. If the argument `locations` is not provided, all objects will be stored in this group.

Returns: Self, invisibly.

Method `detach()`**:** Detach the latitude and longitude from an underlying netCDF resource.

Usage:

```
CFAuxiliaryLongLat$detach()
```

Returns: Self, invisibly.

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 classes

`ncdfCF::CFObject` -> `ncdfCF::CFData` -> `CFAxis`

Active bindings

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

`dimid` The netCDF dimension id of this axis. Setting this value to anything other than the correct value will lead to disaster.

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

`orientation` Set or retrieve the orientation of the axis, a single character with a value of "X", "Y", "Z", "T". Setting the orientation of the axis should only be done when the current orientation is unknown. Setting a wrong value may give unexpected errors in processing of data variables.

`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.

`bounds` Set or retrieve the bounds of this axis as a CFBounds object. When setting the bounds, the bounds values must agree with the coordinates of this 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` (read-only) 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 of this axis. Following elements refer to auxiliary coordinates.

`coordinate_range` (read-only) Retrieve the range of the coordinates of the axis as a vector of two values. The mode of the result depends on the sub-type of the axis.

`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` Logical to indicate if the axis is unlimited. The setting can only be changed if the axis has not yet been written to file.

`time` (read-only) Retrieve the `CFTime` object associated with the axis. Always returns `NULL` but `CFAxisTime` overrides this field.

`is_parametric` (read-only) Logical flag that indicates if the axis has parametric coordinates. Always `FALSE` for all axes except for CFAxisVertical which overrides this method.

Methods

Public methods:

- `CFAxis$new()`
- `CFAxis$print()`
- `CFAxis$brief()`
- `CFAxis$shard()`
- `CFAxis$peek()`
- `CFAxis$detach()`
- `CFAxis$copy_terms()`
- `CFAxis$configure_terms()`
- `CFAxis$identical()`

- CFAxis$can_append()
- CFAxis$copy()
- CFAxis$copy_with_values()
- CFAxis$subset()
- CFAxis$indexOf()
- CFAxis$attach_to_group()
- CFAxis$write()

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(
  var,
  group,
  values,
  start = 1L,
  count = NA,
  orientation = "",
  attributes = data.frame()
)
```

Arguments:

`var` The name of the axis when creating a new axis. When reading an axis from file, the NCVariable object that describes this instance.

`group` The CFGroup that this instance will live in.

`values` Optional. The values of the axis in a vector. Ignored when argument `var` is a `NCVariable` object.

`start` Optional. Integer index where to start reading axis data from file. The index may be `NA` to start reading data from the start.

`count` Optional. Number of elements to read from file. This may be `NA` to read to the end of the data.

`orientation` Optional. The orientation of the axis: "X", "Y", "Z" "T", or "" (default) when not known or relevant.

`attributes` Optional. A `data.frame` with the attributes of the axis. When an empty `data.frame` (default) and argument `var` is an `NCVariable` instance, attributes of the axis will be taken from the netCDF resource.

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()
```

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

Method `detach()`**:** Detach the axis from its underlying netCDF resource, including any dependent CF objects.

Usage:

```
CFAxis$detach()
```

Returns: Self, invisibly.

Method `copy_terms()`**:** Copy the parametric terms of a vertical axis. This method is only useful for `CFAxisVertical` instances having a parametric formulation. This stub is here to make the call to this method succeed with no result for the other descendant classes.

Usage:

```
CFAxis$copy_terms(from, original_axes, new_axes)
```

Arguments:

`from` A CFAxisVertical instance that will receive references to the parametric terms.

`original_axes` List of `CFAxis` instances from the CF object that these parametric terms are copied from.

`new_axes` List of `CFAxis` instances to use with the formula term objects.

Returns: `NULL`

Method `configure_terms()`**:** Configure the function terms of a parametric vertical axis. This method is only useful for `CFAxisVertical` instances having a parametric formulation. This stub is here to make the call to this method succeed with no result for the other descendant classes.

Usage:

```
CFAxis$configure_terms(axes)
```

Arguments:

`axes` List of `CFAxis` instances.

Returns: `NULL`

Method `identical()`**:** Tests if the axis passed to this method is identical to `self`. This only tests for generic properties - class, length, name and attributes - with further assessment done in sub-classes.

Usage:

```
CFAxis$identical(axis, with_attributes = FALSE)
```

Arguments:

`axis` The `CFAxis` instance to test.

`with_attributes` Logical that indicates if the attributes are assessed for equality too. Default is `FALSE`.

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 `copy()`**:** Create a copy of this axis. 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$copy(name = "", group)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: `NULL`

Method `copy_with_values()`**:** Create a copy of this axis but using the supplied values. 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$copy_with_values(name = "", group, values)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`values` The values to the used with the copy of this axis.

Returns: `NULL`

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 `self`. 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(name = "", group, rng = NULL)
```

Arguments:

`name` The name for the new axis if the `rng` argument is provided.

`group` The CFGroup where the copy of this axis will live.

`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 a copy of the 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 `attach_to_group()`**:** Attach this axis to a group. If there is another object with the same name in this group an error is thrown. For associated objects (such as bounds, etc), if another object with the same name is otherwise identical to the associated object then that object will be linked from the variable, otherwise an error is thrown.

Usage:

```
CFAxis$attach_to_group(grp, locations = list())
```

Arguments:

`grp` An instance of CFGroup.

`locations` Optional. A `list` whose named elements correspond to the names of objects associated with this axis, possibly including the axis itself. Each list element has a single character string indicating the group in the hierarchy where the object should be stored. As an example, if the variable has axes "lon" and "lat" and they should be stored in the parent group of `grp`, then specify `locations = list(lon = "..", lat = "..")`. Locations can use absolute paths or relative paths from group `grp`. The axis and associated objects that are not in the list will be stored in group `grp`. If the argument `locations` is not provided, all associated objects will be stored in this group.

Returns: Self, invisibly.

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

Usage:

```
CFAxis$write()
```

Returns: Self, invisibly.

`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::CFData` -> `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$copy()`
- `CFAxisCharacter$copy_with_values()`
- `CFAxisCharacter$slice()`
- `CFAxisCharacter$subset()`
- `CFAxisCharacter$identical()`
- `CFAxisCharacter$append()`
- `CFAxisCharacter$indexOf()`

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

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

Usage:

```
CFAxisCharacter$new(
  var,
  group,
  values,
  start = 1L,
  count = NA,
  attributes = data.frame()
)
```

Arguments:

`var` The name of the axis when creating a new axis. When reading an axis from file, the NCVariable object that describes this instance.

`group` The CFGroup that this instance will live in.

`values` Optional. The values of the axis in a vector. These must be character values. Ignored when argument `var` is a NCVariable object.

`start` Optional. Integer index where to start reading axis data from file. The index may be `NA` to start reading data from the start.

`count` Optional. Number of elements to read from file. This may be `NA` to read to the end of the data.

`attributes` Optional. A `data.frame` with the attributes of the axis. When an empty `data.frame` (default) and argument `var` is an NCVariable instance, attributes of the axis will be taken from the netCDF resource.

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

Usage:

```
CFAxisCharacter$brief()
```

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

Method `copy()`**:** Create a copy of this axis. The copy is completely separate from this axis, meaning that the new axis and all of its components are made from new instances. If this axis is backed by a netCDF resource, the copy will retain the reference to the resource.

Usage:

```
CFAxisCharacter$copy(name = "", group)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created axis.

Method `copy_with_values()`**:** Create a copy of this axis but using the supplied values. The attributes are copied to the new axis. Boundary values and auxiliary coordinates are not copied.

After this operation the attributes of the newly created axes may not be accurate, except for the "actual_range" attribute. The calling code should set, modify or delete attributes as appropriate.

Usage:

```
CFAxisCharacter$copy_with_values(name = "", group, values)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`values` The values to the used with the copy of this axis.

Returns: The newly created axis.

Method `slice()`**:** Given a range of domain coordinate values, returns the indices into the axis that fall within the supplied range.

Usage:

```
CFAxisCharacter$slice(rng)
```

Arguments:

`rng` A character vector whose extreme (alphabetic) values indicate the indices of coordinates to return.

Returns: An integer vector of length 2 with the lower and higher indices into the axis that fall within the range of coordinates in argument `rng`. Returns `NULL` if no values of the axis fall within the range of coordinates.

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:

```
CFAxisCharacter$subset(name = "", group, rng = NULL)
```

Arguments:

`name` The name for the new axis. If an empty string is passed (default), will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`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 a copy of the axis.

Returns: A new `CFAxisCharacter` instance covering the indicated range of indices. If the value of the argument `rng` is `NULL`, return a copy of this axis as the new 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, group)
```

Arguments:

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

`group` The CFGroup where the copy of this axis will live.

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`.

`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::CFData` -> `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$brief()`
- `CFAxisDiscrete$copy()`
- `CFAxisDiscrete$indexOf()`
- `CFAxisDiscrete$slice()`
- `CFAxisDiscrete$subset()`
- `CFAxisDiscrete$append()`
- `CFAxisDiscrete$write()`

Method `new()`**:** Create a new instance of this class. The values of this axis are always a sequence, but the sequence may start with any positive value such that the length of this instance falls within the length of the axis on file, if there is one.

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

Usage:

```
CFAxisDiscrete$new(var, group, start = 1L, count)
```

Arguments:

`var` The name of the axis when creating a new axis. When reading an axis from file, the NCVariable object that describes this instance.

`group` The CFGroup that this instance will live in.

`start` Optional. Integer value that indicates the starting value of this axis. Defults to `1L`.

`count` Number of elements in the axis.

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 `copy()`**:** Create a copy of this axis. The copy is completely separate from this axis, meaning that both this axis and all of its components are made from new instances.

Usage:

```
CFAxisDiscrete$copy(name = "", group)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created 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. 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 `slice()`**:** Given a range of coordinate values, returns the indices into the axis that fall within the supplied range. If the axis has auxiliary coordinates selected then these will be used for the identification of the indices to return.

Usage:

```
CFAxisDiscrete$slice(rng)
```

Arguments:

`rng` A vector whose extreme values indicate the indices of coordinates to return. The mode of the vector has to be integer or agree with any auxiliary coordinates selected.

Returns: An integer vector of length 2 with the lower and higher indices into the axis that fall within the range of coordinates in argument `rng`. Returns `NULL` if no (boundary) values of the axis fall within the range of coordinates.

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(name = "", group, rng = NULL)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`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 a copy of the axis.

Returns: A new `CFAxisDiscrete` instance covering the indicated range of indices. If the value of the argument is `NULL`, return a copy of `self` as the new axis.

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

Usage:

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

Arguments:

`from` An instance of `CFAxisDiscrete` whose length to add to this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: A new `CFAxisDiscrete` instance with a length that is the sum of the lengths of this axis and the `from` axis.

Method `write()`**:** Write the axis to a netCDF file. A discrete axis does not have any attributes or values to write.

Usage:

```
CFAxisDiscrete$write()
```

Returns: Self, invisibly.

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::CFData -> ncdfCF::CFAxis -> ncdfCF::CFAxisNumeric -> CFAxisLatitude

Active bindings

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

Methods

Public methods:

- CFAxisLatitude$new()
- CFAxisLatitude$copy()
- CFAxisLatitude$copy_with_values()
- CFAxisLatitude$subset()
- CFAxisLatitude$append()

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(
  var,
  group,
  values,
  start = 1L,
  count = NA,
  attributes = data.frame()
)
```

Arguments:

var The name of the axis when creating a new axis. When reading an axis from file, the NCVariable object that describes this instance.

group The CFGroup that this instance will live in.

values Optional. The values of the axis in a vector. The values have to be numeric within the range (-90, 90) and monotonic. Ignored when argument var is a NCVariable object.

start Optional. Integer index where to start reading axis data from file. The index may be NA to start reading data from the start.

`count` Optional. Number of elements to read from file. This may be `NA` to read to the end of the data.

`attributes` Optional. A `data.frame` with the attributes of the axis. When an empty `data.frame` (default) and argument `var` is an NCVariable instance, attributes of the axis will be taken from the netCDF resource.

Method `copy()`**:** Create a copy of this axis. The copy is completely separate from `self`, meaning that both `self` and all of its components are made from new instances.

Usage:

```
CFAxisLatitude$copy(name = "", group)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created axis.

Method `copy_with_values()`**:** Create a copy of this axis but using the supplied values. The attributes are copied to the new axis. Boundary values and auxiliary coordinates are not copied. After this operation the attributes of the newly created axes may not be accurate, except for the "actual_range" attribute. The calling code should set, modify or delete attributes as appropriate.

Usage:

```
CFAxisLatitude$copy_with_values(name = "", group, values)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`values` The values to the used with the copy of this axis.

Returns: The newly created axis.

Method `subset()`**:** Return a latitude 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(name = "", group, rng = NULL)
```

Arguments:

`name` The name for the new axis. If an empty string is passed (default), will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`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 a copy of the axis.

Returns: A new `CFAxisLatitude` instance covering the indicated range of indices. If the value of the argument `rng` is `NULL`, return a copy of `self` as the new axis.

Method `append()`**:** Append a vector of values at the end of the current values of the axis. Boundary values are appended as well but if either this axis or the `from` axis does not have boundary values, neither will the resulting axis.

Usage:

```
CFAxisLatitude$append(from)
```

Arguments:

`from` An instance of `CFAxisLatitude` whose values to append to the values of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: A new `CFAxisLatitude` instance with values from this axis and the `from` axis appended.

`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::CFData` -> `ncdfCF::CFAxis` -> `ncdfCF::CFAxisNumeric` -> `CFAxisLongitude`

Active bindings

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

Methods

Public methods:

- `CFAxisLongitude$new()`
- `CFAxisLongitude$copy()`
- `CFAxisLongitude$copy_with_values()`
- `CFAxisLongitude$subset()`
- `CFAxisLongitude$append()`

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(
  var,
  group,
  values,
  start = 1L,
  count = NA,
  attributes = data.frame()
)
```

Arguments:

`var` The name of the axis when creating a new axis. When reading an axis from file, the NCVariable object that describes this instance.

`group` The CFGroup that this instance will live in.

`values` Optional. The values of the axis in a vector. The values have to be numeric with the maximum value no larger than the minimum value + 360, and monotonic. Ignored when argument `var` is a NCVariable object.

`start` Optional. Integer index where to start reading axis data from file. The index may be `NA` to start reading data from the start.

`count` Optional. Number of elements to read from file. This may be `NA` to read to the end of the data.

`attributes` Optional. A `data.frame` with the attributes of the axis. When an empty `data.frame` (default) and argument `var` is an NCVariable instance, attributes of the axis will be taken from the netCDF resource.

Method `copy()`**:** Create a copy of this axis. The copy is completely separate from `self`, meaning that both `self` and all of its components are made from new instances.

Usage:

```
CFAxisLongitude$copy(name = "", group)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created axis.

Method `copy_with_values()`**:** Create a copy of this axis but using the supplied values. The attributes are copied to the new axis. Boundary values and auxiliary coordinates are not copied. After this operation the attributes of the newly created axes may not be accurate, except for the "actual_range" attribute. The calling code should set, modify or delete attributes as appropriate.

Usage:

```
CFAxisLongitude$copy_with_values(name = "", group, values)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`values` The values to the used with the copy of this axis.

Returns: The newly created axis.

Method `subset()`**:** Return a longitude 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(name = "", group, rng = NULL)
```

Arguments:

`name` The name for the new axis. If an empty string is passed (default), will use the name of this axis.

group The CFGroup where the copy of this axis will live.

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 a copy of the axis.

Returns: A new CFAxisLongitude instance covering the indicated range of indices. If the value of the argument rng is NULL, return a copy of self as the new axis.

Method append(): Append a vector of values at the end of the current values of the axis. Boundary values are appended as well but if either this axis or the from axis does not have boundary values, neither will the resulting axis.

Usage:

```
CFAxisLongitude$append(from)
```

Arguments:

from An instance of CFAxisLongitude whose values to append to the values of this axis.

group The CFGroup where the copy of this axis will live.

Returns: A new CFAxisLongitude instance with values from this axis and the from axis appended.

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::CFData -> 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$indexOf()
- CFAxisNumeric$slice()

- CFAxisNumeric$copy()
- CFAxisNumeric$copy_with_values()
- CFAxisNumeric$identical()
- CFAxisNumeric$append()
- CFAxisNumeric$subset()

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

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

Usage:

```
CFAxisNumeric$new(
  var,
  group,
  values,
  start = 1L,
  count = NA,
  orientation = "",
  attributes = data.frame()
)
```

Arguments:

`var` The name of the axis when creating a new axis. When reading an axis from file, the NCVariable object that describes this instance.

`group` The CFGroup that this instance will live in.

`values` Optional. The values of the axis in a vector. The values have to be numeric with the maximum value no larger than the minimum value + 360, and monotonic. Ignored when argument `var` is a `NCVariable` object.

`start` Optional. Integer index where to start reading axis data from file. The index may be `NA` to start reading data from the start.

`count` Optional. Number of elements to read from file. This may be `NA` to read to the end of the data.

`orientation` Optional. The orientation of the axis: "X", "Y", "Z" "T", or "" (default) when not known or relevant.

`attributes` Optional. A `data.frame` with the attributes of the axis. When an empty `data.frame` (default) and argument `var` is an `NCVariable` instance, attributes of the axis will be taken from the netCDF resource.

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 boundary values then the supplied coordinates must fall within the boundaries of an axis coordinate 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, 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`. If the axis has boundary values, then values of `x` that do not fall on or between the boundaries of an axis coordinate are returned as `NA`.

Method `slice()`**:** Given a range of domain coordinate values, returns the indices into the axis that fall within the supplied range. If the axis has bounds, any coordinate whose boundary values fall entirely or partially within the supplied range will be included in the result.

Usage:

```
CFAxisNumeric$slice(rng)
```

Arguments:

`rng` A numeric vector whose extreme values indicate the indices of coordinates to return.

Returns: An integer vector of length 2 with the lower and higher indices into the axis that fall within the range of coordinates in argument `rng`. Returns `NULL` if no (boundary) values of the axis fall within the range of coordinates.

Method `copy()`**:** Create a copy of this axis. The copy is completely separate from `self`, meaning that both `self` and all of its components are made from new instances.

Usage:

```
CFAxisNumeric$copy(name = "", group)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created axis.

Method `copy_with_values()`**:** Create a copy of this axis but using the supplied values. The attributes are copied to the new axis. Boundary values and auxiliary coordinates are not copied. After this operation the attributes of the newly created axes may not be accurate, except for the "actual_range" attribute. The calling code should set, modify or delete attributes as appropriate.

Usage:

```
CFAxisNumeric$copy_with_values(name = "", group, values)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`values` The values to the used with the copy of this axis.

Returns: The newly created axis.

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. Boundary values are appended as well but if either this axis or the `from` axis does not have boundary values, neither will the resulting axis.

Usage:

```
CFAxisNumeric$append(from, group)
```

Arguments:

`from` An instance of `CFAxisNumeric` whose values to append to the values of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: A new `CFAxisNumeric` instance with values from this axis 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(name = "", group, rng = NULL)
```

Arguments:

`name` The name for the new axis. If an empty string is passed (default), will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`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 a copy of the axis.

Returns: A new `CFAxisNumeric` instance covering the indicated range of indices. If the value of the argument `rng` is `NULL`, return a copy of this axis as the new axis.

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::CFData -> ncdfCF::CFAxis -> CFAxisTime

Active bindings

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

time (read-only) Retrieve the CFTime instance that manages the values of this axis.

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

Methods

Public methods:

- CFAxisTime$new()
- CFAxisTime$print()
- CFAxisTime$brief()
- CFAxisTime$identical()
- CFAxisTime$copy()
- CFAxisTime$copy_with_values()
- CFAxisTime$append()
- CFAxisTime$indexOf()
- CFAxisTime$slice()
- CFAxisTime$subset()

Method new(): Create a new instance of this class, including its boundary values. A CFTime or CFClimatology instance will also be created to manage the time magic.

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

Usage:

```
CFAxisTime$new(
  var,
  group,
  values,
  start = 1L,
  count = NA,
  attributes = data.frame()
)
```

Arguments:

`var` The name of the axis when creating a new axis. When reading an axis from file, the NCVariable object that describes this instance.

`group` The CFGroup that this instance will live in.

`values` Either the numeric values of this axis, in which case argument `var` must be a `NCVariable`, or an instance of `CFTime` or `CFClimatology` with bounds set, and then argument `var` must be a name for the axis.

`start` Optional. Integer index where to start reading axis data from file. The index may be `NA` to start reading data from the start.

`count` Optional. Number of elements to read from file. This may be `NA` to read to the end of the data.

`attributes` Optional. A `data.frame` with the attributes of the axis. When an empty `data.frame` (default) and argument `var` is an NCVariable instance, attributes of the axis will be taken from the netCDF resource.

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.

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

Usage:

```
CFAxisTime$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:

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

Arguments:

`axis` The `CFAxisTime` instance to test.

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

Method `copy()`**:** Create a copy of this axis. The copy is completely separate from `self`, meaning that both `self` and all of its components are made from new instances.

Usage:

```
CFAxisTime$copy(name = "", group)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created axis.

Method `copy_with_values()`**:** Create a copy of this axis but using the supplied values. The attributes are copied to the new axis. Boundary values and auxiliary coordinates are not copied. After this operation the attributes of the newly created axes may not be accurate, except for the "actual_range" attribute. The calling code should set, modify or delete attributes as appropriate.

Usage:

```
CFAxisTime$copy_with_values(name = "", group, values)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`values` The values to the used with the copy of this axis. This can be a `CFTime` instance, a vector of numeric values, a vector of character timestamps in ISO8601 or UDUNITS format, or a vector of `POSIXct` or `Date` values. If not a `CFTime` instance, the `values` will be converted into a `CFTime` instance using the definition and calendar of this axis.

Returns: The newly created axis.

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

Usage:

```
CFAxisTime$append(from, group)
```

Arguments:

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

`group` The CFGroup where the copy of this axis will live.

Returns: A new `CFAxisTime` instance with values from this axis 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: A 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 `NULL` 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(name = "", group, rng = NULL)
```

Arguments:

`name` The name for the new axis. If an empty string is passed (default), will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`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 a copy of the axis.

Returns: A new `CFAxisTime` instance covering the indicated range of indices. If the value of the argument `rng` is `NULL`, return a copy of `self` as the new axis.

`CFAxisVertical` *Vertical CF axis object*

Description

This class represents a vertical axis, which may be parametric. A regular vertical axis behaves like any other numeric axis. A parametric vertical axis, on the other hand, is defined through an index value that is contained in the axis coordinates, with additional data variables that hold ancillary "formula terms" with which to calculate physical axis coordinates. It is used in atmosphere and ocean data sets.

Parametric vertical axes can only be read from file, not created from scratch.

Super classes

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

Active bindings

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

`formula_terms` (read-only) A `data.frame` with the "formula_terms" to calculate the parametric axis values.

`is_parametric` (read-only) Logical flag that indicates if the coordinates of the axis are parametric.

`parametric_coordinates` (read-only) Retrieve the parametric coordinates of this vertical axis as a CFVariable.

`computed_name` (read-only) The name of the computed parameterised coordinates. If the parameterised coordinates have not been computed yet the computed name is an empty string.

`computed_units` (read-only) Return the units of the computed parameterised coordinates, if computed, otherwise return `NULL`. This will access the standard names table.

Methods

Public methods:

- CFAxisVertical$new()
- CFAxisVertical$attach_to_group()
- CFAxisVertical$detach()
- CFAxisVertical$copy()
- CFAxisVertical$copy_with_values()
- CFAxisVertical$set_parametric_terms()
- CFAxisVertical$append()
- CFAxisVertical$subset()
- CFAxisVertical$subset_parametric_terms()

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

Usage:

```
CFAxisVertical$new(
  var,
  group,
  values,
  start = 1L,
  count = NA,
  attributes = data.frame()
)
```

Arguments:

`var` The name of the axis when creating a new axis. When reading an axis from file, the NCVariable object that describes this instance.

`group` The CFGroup that this instance will live in.

`values` Optional. The values of the axis in a vector. The values have to be numeric and monotonic.

`start` Optional. Integer index where to start reading axis data from file. The index may be `NA` to start reading data from the start.

`count` Optional. Number of elements to read from file. This may be `NA` to read to the end of the data.

`attributes` Optional. A `data.frame` with the attributes of the axis. When an empty `data.frame` (default) and argument `var` is an `NCVariable` instance, attributes of the axis will be taken from the netCDF resource.

Method `attach_to_group()`**:** Attach this verical axis to a group, including any parameteric terms. If there is another object with the same name in this group an error is thrown. For associated objects (such as bounds, etc), if another object with the same name is otherwise identical to the associated object then that object will be linked from the variable, otherwise an error is thrown.

Usage:

```
CFAxisVertical$attach_to_group(grp, locations = list())
```

Arguments:

`grp` An instance of CFGroup.

`locations` Optional. A `list` whose named elements correspond to the names of objects associated with this axis, possibly including the axis itself. Each list element has a single character string indicating the group in the hierarchy where the object should be stored. As an example, if the variable has axes "lon" and "lat" and they should be stored in the parent group of `grp`, then specify `locations = list(lon = "..", lat = "..")`. Locations can use absolute paths or relative paths from group `grp`. The axis and associated objects that are not in the list will be stored in group `grp`. If the argument `locations` is not provided, all associated objects will be stored in this group.

Returns: Self, invisibly.

Method `detach()`**:** Detach the parametric terms from an underlying netCDF resource.

Usage:

```
CFAxisVertical$detach()
```

Returns: Self, invisibly.

Method `copy()`**:** Create a copy of this axis. The copy is completely separate from this instance, meaning that the copies of both this instance and all of its components are made as new instances.

Usage:

```
CFAxisVertical$copy(name = "", group)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created axis.

Method `copy_with_values()`**:** Create a copy of this axis but using the supplied values. The attributes are copied to the new axis. Boundary values, parametric coordinates and auxiliary coordinates are not copied.

After this operation the attributes of the newly created axes may not be accurate, except for the "actual_range" attribute. The calling code should set, modify or delete attributes as appropriate.

Usage:

```
CFAxisVertical$copy_with_values(name = "", group, values)
```

Arguments:

`name` The name for the new axis. If an empty string is passed, will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`values` The values to the used with the copy of this axis.

Returns: The newly created axis.

Method `set_parametric_terms()`**:** Set the parametric terms for this axis. The name and the terms have to fully describe a CF parametric vertical axis.

The terms must also agree with the other axes used by any data variable that refers to this axis. That is not checked here so the calling code must make that assertion.

Usage:

```
CFAxisVertical$set_parametric_terms(sn, terms)
```

Arguments:

`sn` The "standard_name" of the parametric formulation. See the CF documentation for details.

`terms` A `data.frame` with columns `term`, `variable` and `param` containing the terms of the formula to calculate the axis values. Column `param` has the references to the variables that hold the data for each term.

Method `append()`**:** Append a vector of values at the end of the current values of the axis. Boundary values are appended as well but if either this axis or the `from` axis does not have boundary values, neither will the resulting axis.

This method is not recommended for parametric vertical axes. Any parametric terms will be deleted. If appending of parametric axes is required, the calling code should first read out the parametric terms and merge them with the parametric terms of the `from` axis before setting them back for this axis.

Usage:

```
CFAxisVertical$append(from)
```

Arguments:

`from` An instance of `CFAxisVertical` whose values to append to the values of this axis.

`group` The CFGroup where the copy of this axis will live.

Returns: A new `CFAxisVertical` instance with values from this axis 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. If this axis has parametric terms, these are not subset here - they should be separately treated once all associated axes in the terms have been subset. That happens automatically in `CFVariable` methods which call the `subset_parametric_terms()` method.

Usage:

```
CFAxisVertical$subset(name = "", group, rng = NULL)
```

Arguments:

`name` The name for the new axis. If an empty string is passed (default), will use the name of this axis.

`group` The CFGroup where the copy of this axis will live.

`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 a copy of the axis.

Returns: A new `CFAxisVertical` instance covering the indicated range of indices. If the value of the argument `rng` is `NULL`, return a copy of this axis as the new axis.

Method `subset_parametric_terms()`**:** Subset the parametric terms of this axis.

Usage:

```
CFAxisVertical$subset_parametric_terms(
  original_axis_names,
  new_axes,
  start,
  count,
  aux = NULL,
  ZT_dim = NULL
)
```

Arguments:

`original_axis_names` Character vector of names of the axes prior to a modifying operation in the owning data variable

`new_axes` List of `CFAxis` instances to use for the subsetting.

`start` The indices to start reading data from the file, as an integer vector at least as long as the number of axis for the term.

`count` The number of values to read from the file, as an integer vector at least as long as the number of axis for the term.

`aux` Optional. List with the parameters for an auxiliary grid transformation. Default is `NULL`.

`ZT_dim` Optional. Dimensions of the non-grid axes when an auxiliary grid transformation is specified.

Returns: Self, invisibly. The parametric terms will have been subset in this axis.

References

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

---

`CFBounds` *CF boundary variable*

Description

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

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

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFData` -> `CFBounds`

Active bindings

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

`length` (read-only) The length of the second dimension of the data, i.e. the number of boundary values.

`vertices` (read-only) The length of the first dimension of the data, i.e. the number of vertices that make up a boundary.

`values` Set or retrieve the boundary values of this object. Upon retrieval, values are read from the netCDF resource, if there is one, upon first access and cached thereafter. Upon setting values, if there is a linked netCDF resource, this object will be detached from it.

Methods

Public methods:

- CFBounds$new()
- CFBounds$print()
- CFBounds$print_boundary_values()
- CFBounds$range()
- CFBounds$copy()
- CFBounds$subset()
- CFBounds$append()
- CFBounds$write()

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

Usage:

```
CFBounds$new(
  var,
  group,
  values,
  start = NA,
  count = NA,
  attributes = data.frame(),
  owner_dims = 1L
)
```

Arguments:

`var` The name of the boundary variable when creating a new boundary variable. When reading a boundary variable from file, the NCVariable object that describes this instance.

`group` The CFGroup that this instance will live in.

`values` Optional. The values of the boundary variable. This must be a numeric matrix whose first dimension has a length equal to the number of vertices for each boundary, and the second dimension is as long as the `CFObject` instances that use these boundary values. Ignored when argument `var` is a `NCVariable` object.

`start` Optional. Vector of indices where to start reading boundary data along the dimensions of the data. The vector must be `NA` to read all data, otherwise it must have a length equal to the dimensionality of the owning object + 1.

`count` Optional. Vector of number of elements to read along each dimension of the boundary data. The vector must be `NA` to read to the end of each dimension, otherwise it must have a length equal to the dimensionality of the owning object + 1.

`attributes` Optional. A `data.frame` with the attributes of the boundary object. When an empty `data.frame` (default) and argument `var` is an `NCVariable` instance, attributes of the bounds object will be taken from the netCDF resource.

`owner_dims` Optional, the number of dimensions of the object that these boundary values pertain to. Default is 1.

Returns: A new instance of this class.

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

Usage:

```
CFBounds$print(attributes = TRUE, ...)
```

Arguments:

`attributes` Default `TRUE`, flag to indicate if the attributes of the boundary values should be printed.

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

Method `print_boundary_values()`**:** Print the boundary values to the console. This method is not very useful to call directly - instead, call `$print()`, which will call this method. These boundary values are also printed when printing an axis that has boundary values associated with it.

Usage:

```
CFBounds$print_boundary_values()
```

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

Usage:

```
CFBounds$range()
```

Method `copy()`**:** Create a copy of this bounds object The copy is completely separate from `self`, meaning that both `self` and all of its components are made from new instances.

Usage:

```
CFBounds$copy(name = "", group)
```

Arguments:

`name` The name for the new bounds object. If an empty string is passed, will use the name of this bounds object.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created bounds object.

Method `subset()`**:** Return a boundary variable spanning a smaller coordinate range. This currently only applies to 1-D axes.

This method returns boundary values which span the range of indices given by the `rng` argument.

Usage:

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

Arguments:

`group` The CFGroup where the copy of these bounds will live.

`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 `append()`**:** Append boundary values at the end of the current values of the boundary variable.

Usage:

```
CFBounds$append(from, group)
```

Arguments:

`from` An instance of `CFBounds` whose values to append to the values of this boundary variable.

`group` The CFGroup where the copy of these bounds will live.

Returns: A new `CFBounds` instance with values from this boundary variable and the `from` boundary variable appended. If argument `from` is `NULL`, return `NULL`.

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

Usage:

```
CFBounds$write(object_id)
```

Arguments:

`object_id` The integer dimid of the object that uses these boundary values, usually an axis but could also be an auxiliary CV.

`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.

Active bindings

`measure` (read-only) Retrieve 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$link()`
- `CFCellMeasure$detach()`
- `CFCellMeasure$clone()`

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

Usage:

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

Arguments:

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

`name` The name of the cell measure variable. Ignored 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 CFVariable 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 `detach()`**:** Detach the internal data variable from an underlying netCDF resource.

Usage:

```
CFCellMeasure$detach()
```

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.

`CFData` *CF data object*

Description

This class is a basic ancestor to all classes that contain data from a netCDF resource, specifically data variables and axes. More useful classes use this class as ancestor.

Super class

`ncdfCF::CFObject` -> `CFData`

Active bindings

`data_type` Set or retrieve the data type of the data in the object. Setting the data type to a wrong value can have unpredictable and mostly catastrophic consequences.

`ndims` (read-only) Retrieve the dimensionality of the data in the array.

`NC_map` Returns a list with columns "start" and "count" giving the indices for reading the data of this object from a netCDF resource. The list is empty if this object is not backed by a netCDF resource.

Methods

Public methods:

- `CFData$new()`
- `CFData$detach()`
- `CFData$dim()`
- `CFData$read_data()`
- `CFData$read_chunk()`

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

Usage:

```
CFData$new(
  obj,
  group,
  values,
  start = 1L,
  count = NA,
  attributes = data.frame()
)
```

Arguments:

`obj` The NCVariable instance upon which this CF object is based when read from a netCDF resource, or the name for the new CF object to be created.

`group` The CFGroup that this instance will live in.

`values` Optional. The values of the object in an array. Ignored when argument `obj` is an `NCVariable` instance.

`start` Optional. Vector of indices where to start reading data along the dimensions of the array on file. The vector must be `NA` to read all data, otherwise it must have agree with the dimensions of the array on file. Default value is `1`, i.e. start from the beginning of the 1-D NC variable. Ignored when argument `obj` is not an `NCVariable` instance.

`count` Optional. Vector of number of elements to read along each dimension of the array on file. The vector must be `NA` to read to the end of each dimension, otherwise its value must agree with the corresponding `start` value and the dimension of the array on file. Default is `NA`. Ignored when argument `obj` is not an `NCVariable` instance.

`attributes` Optional. A `data.frame` with the attributes of the object.

Returns: A `CFData` instance.

Method `detach()`**:** Detach the current object from its underlying netCDF resource. If necessary, data is read from the resource before detaching.

Usage:

```
CFData$detach()
```

Method `dim()`**:** Retrieve the dimensions of the data of this object.

Usage:

```
CFData$dim(dimension)
```

Arguments:

`dimension` Optional. The index of the dimension to retrieve the length for. If omitted, retrieve the lengths of all dimensions.

Returns: Integer vector with the length of each requested dimension. Read the data of the CF object from the NC file. The data is cached by `self` so repeated calls do not access the netCDF resource, unless argument `refresh` is `TRUE`. This method will not assess how big the data is before reading it so there is a chance that memory will be exhausted. The calling code should check for this possibility and break up the reading of data into chunks.

Method `read_data()`**:**

Usage:

```
CFData$read_data(refresh = FALSE)
```

Arguments:

`refresh` Should the data be read from file if the object is linked? This will replace current values, if previously loaded. Default `FALSE`.

Returns: An array of data, invisibly, as prescribed by the `start` and `count` values used to create this object. If the object is not backed by a netCDF resource, returns `NULL`. Read a chunk of raw data of the CF object, as defined by the `start` and `count` vectors. Note that these vectors are relative to any subset of the data variable that this CF object refers to. The data read by this method will not be stored in `self` so the calling code must take a reference to it.

Method `read_chunk()`**:**

Usage:

```
CFData$read_chunk(start, count)
```

Arguments:

`start` Vector of indices where to start reading data along the dimensions of the array. The vector must be `NA` to read all data, otherwise it must agree with the dimensions of the array.

`count` Vector of number of elements to read along each dimension of the array. The vector must be `NA` to read to the end of each dimension, otherwise its value must agree with the corresponding `start` value and the dimension of the array.

Returns: An array of data, as prescribed by the `start` and `count` arguments, or `NULL` if there is no data.

`CFDataset` *CF data set*

Description

This class represents a CF data set, the object that encapsulates a netCDF resource. You should never instantiate this class directly; instead, call `open_ncdf()` which will return an instance that has all properties read from the netCDF resource, or `create_ncdf()` for a new, empty instance. 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).

`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$set_attribute()`
- `CFDataset$append_attribute()`
- `CFDataset$delete_attribute()`
- `CFDataset$add_variable()`
- `CFDataset$save()`
- `CFDataset$clone()`

Method `new()`**:** Create an instance of this class. Do not instantiate this class directly; instead, call `open_ncdf()` which will return an instance that has all properties read from the netCDF resource, or `create_ncdf()` for a new, empty instance.

Usage:

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

Arguments:

`resource` An instance of `NCResource` that links to the netCDF resource, or a character string with the name of a new data set.

`format` Character string with the format of the netCDF resource as reported by the call opening the resource. Ignored when argument `resource` is a character string.

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)
```

Arguments:

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

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 `set_attribute()`**:** Add an attribute to the global attributes. If an attribute `name` already exists, it will be overwritten.

Usage:

```
CFDataset$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 a global 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:

```
CFDataset$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:

```
CFDataset$delete_attribute(name)
```

Arguments:

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

Returns: Self, invisibly.

Method `add_variable()`**:** Add a CFVariable object to the data set. If there is another object with the same name in the group where the data variable should be placed an error is thrown. For objects associated with the data variable (such as axes, CRS, boundary variables, etc), if another object with the same name is otherwise identical to the associated object then that object will be linked from the variable, otherwise an error is thrown.

Usage:

```
CFDataset$add_variable(var, group, locations = list())
```

Arguments:

`var` An instance of `CFVariable` or any of its descendants.

`group` Optional. An instance of CFGroup where the data variable should be located. If omitted, the data variable will be stored in the root group.

`locations` Optional. A `list` whose named elements correspond to the names of objects associated with the data variable in argument `var`. Each list element has a single character string indicating the group in the hierarchy where the object should be stored. As an example, if the data variable has axes "lon" and "lat" and they should be stored in the parent group of `group`, then specify `locations = list(lon = "..", lat = "..")`. Locations can use absolute paths or relative paths from the `group`. Associated objects that are not in the list will be stored in `group`. If the argument `locations` is not provided, all associated objects will be stored in `group`.

Returns: Argument `var`, invisibly.

Method `save()`**:** Save the data set to file, including its subordinate objects such as attributes, data variables, axes, CRS, etc.

Usage:

```
CFDataset$save(fn = NULL, pack = FALSE)
```

Arguments:

`fn` Optional. Fully-qualified file name indicating where to save the data set to. This argument must be provided if the data set is virtual. If the argument is provided on a data set that was read from a netCDF file and it does not point to that netCDF file, a new netCDF file will be written to the indicated location. If the argument is the same file name as before, the existing netCDF file will be updated.

`pack` Optional. Logical to indicate if the data should be packed; default is `FALSE`. 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:

```
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`

Active bindings

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

Methods

Public methods:

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

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

Note that when a new grid mapping object is created (as opposed to reading from a netCDF resource), only the `grid_mapping_name` attribute will be set. The caller must set all other parameters through their respective attributes, following the CF Metadata Conventions.

Usage:

```
CFGridMapping$new(var, group, grid_mapping_name)
```

Arguments:

`var` When creating a new grid mapping object, the name of the object. When reading from a netCDF resource, the netCDF variable that describes this instance.

`group` The CFGroup that this instance will live in.

`grid_mapping_name` Optional. When creating a new grid mapping object, the formal name of the grid mapping, as specified in the CF Metadata Conventions. This value is stored in the new object as attribute "grid_mapping_name". Ignored when argument `var` is a NC object.

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` 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()
```

Returns: Self, invisibly.

References

https://docs.ogc.org/is/18-010r11/18-010r11.pdf https://cfconventions.org/cf-conventions/cf-conventions.html#appendix-grid-mappings

CFGroup *Group for CF objects*

Description

This class represents a CF group, the object that holds elements like dimensions and variables of a CFDataset.

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. Changing the properties of a group other than its name may very well invalidate the CF objects or even the netCDF file.

Super class

ncdfCF::CFObject -> CFGroup

Active bindings

`parent` (read-only) The parent group of the current group, or its owning data set for the root node.

`name` Set or retrieve the name of the group. Note that the name is always relative to the location in the hierarchy that the group is in and it should thus not be qualified by backslashes. The name has to be a valid CF name. The name of the root group cannot be changed.

`fullname` (read-only) The fully qualified absolute path of the group.

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

`data_set` (read-only) Retrieve the CFDataset that the group belongs to. If the group is not attached to a `CFDataset`, returns `NULL`.

`has_subgroups` (read-only) Does the current group have subgroups?

`subgroups` (read-only) Retrieve the list of the subgroups of the current group.

`CFobjects` (read-only) Retrieve the list of CF objects of the current group.

Methods

Public methods:

- `CFGroup$new()`
- `CFGroup$print()`
- `CFGroup$hierarchy()`
- `CFGroup$subgroup_names()`
- `CFGroup$create_subgroup()`
- `CFGroup$add_subgroups()`
- `CFGroup$add_CF_object()`
- `CFGroup$remove_CF_object()`
- `CFGroup$objects()`
- `CFGroup$find_by_name()`

- CFGroup$add_variable()
- CFGroup$write()
- CFGroup$write_variables()

Method `new()`**:** Create a new CF group instance.

Usage:

```
CFGroup$new(grp, parent)
```

Arguments:

`grp` Either a NCGroup instance when opening a netCDF resource, or a character string with a name for the group when creating a new CF group in memory. When a character string, it should be the local name, without any slash "/" characters. For the root group, specify an empty string "".

`parent` The parent group for this group, or a `CFDataset` for the root group.

Returns: An instance of this class.

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

Usage:

```
CFGroup$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:

```
CFGroup$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 `subgroup_names()`**:** Retrieve the names of the subgroups of the current group.

Usage:

```
CFGroup$subgroup_names(recursive = TRUE)
```

Arguments:

`recursive` Logical, default is `TRUE`. If `TRUE`, include names of recursively through the group hierarchy.

Returns: A character vector with the names of the subgroups of the current group. If `recursive = TRUE`, the names will be fully qualified with their path.

Method `create_subgroup()`**:** Create a new group as a subgroup of the current group.

Usage:

```
CFGroup$create_subgroup(name)
```

Arguments:

`name` The name of the new subgroup. This must be a valid CF name, so not contain any slash '/' characters among other restrictions, and it cannot be already present in the group.

Returns: The newly created group, or an error.

Method `add_subgroups()`**:** Add subgroups to the current group. These subgroups must be fully formed, including having set their parent to this group. Use the `create_subgroup()` method to add a group from scratch.

Usage:

```
CFGroup$add_subgroups(grps)
```

Arguments:

`grps` A `CFGroup`, or `list` thereof.

Returns: Self, invisibly.

Method `add_CF_object()`**:** Add one or more CF objects to the current group. This is an internal method that should not be invoked by the user. The objects to be added are considered atomic and not assessed for any contained objects. Use a method like `add_variable()` to add a CF variable to this group as well as its composing sub-objects such as axes.

Usage:

```
CFGroup$add_CF_object(obj, silent = TRUE)
```

Arguments:

`obj` An instance of a `CFObject` descendant class, or a `list` thereof. If it is a `list`, the list elements must be named after the CF object they contain.

`silent` Logical. If `TRUE` (default), CF objects in argument `obj` whose name is already present in the list of CF objects *and* whose class is identical to the already present object are silently dropped; otherwise or when the argument is `FALSE` an error is thrown.

Returns: Self, invisibly, or an error.

Method `remove_CF_object()`**:** Remove a CF objects from the current group. This is an internal method that should not be invoked by the user. The objects to be removed are considered atomic and not assessed for any contained objects.

Usage:

```
CFGroup$remove_CF_object(obj)
```

Arguments:

`obj` The integer id property of the CF object to remove from this group.

Returns: Self, invisibly.

Method `objects()`**:** This method lists the CF objects of a certain class located in this group, optionally including objects in subgroups.

Usage:

```
CFGroup$objects(cls, recursive = TRUE)
```

Arguments:

`cls` Character vector of classes whose objects to retrieve. Note that subclasses are automatically retrieved as well, so specifying `cls = "CFAxis"` will retrieve all axes defined in this group.

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

Returns: A list of CFObject instances.

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:

```
CFGroup$find_by_name(name)
```

Arguments:

`name` The name of an object, with an optional absolute or relative group path from the calling group. The object must be an CF construct: group, data variable, axis, auxiliary axis, label, grid mapping, etc.

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

Method `add_variable()`**:** Add a CFVariable object to the group. If there is another object with the same name in this group an error is thrown. For associated objects (such as axes, CRS, boundary variables, etc), if another object with the same name is otherwise identical to the associated object then that object will be linked from the variable, otherwise an error is thrown.

Usage:

```
CFGroup$add_variable(var, locations = list())
```

Arguments:

`var` An instance of `CFVariable` or any of its descendants.

`locations` Optional. A `list` whose named elements correspond to the names of objects associated with the variable in argument `var`. Each list element has a single character string indicating the group in the hierarchy where the object should be stored. As an example, if the variable has axes "lon" and "lat" and they should be stored in the parent group of this group, then specify `locations = list(lon = "..", lat = "..")`. Locations can use absolute paths or relative paths from the current group. Associated objects that are not in the list will be stored in this group. If the argument `locations` is not provided, all associated objects will be stored in this group.

Returns: Argument `var`, invisibly.

Method `write()`**:** Write the group to file, including its attributes, if it doesn't already exist.

Usage:

```
CFGroup$write(recursive = TRUE)
```

Arguments:

`recursive` If `TRUE` (default), write sub-groups as well.

Returns: Self, invisibly.

Method `write_variables()`**:** Write data variables in the group to file, including its associated objects, if it doesn't already exist.

Usage:

```
CFGroup$write_variables(pack = FALSE, recursive = TRUE)
```

Arguments:

`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.

`recursive` If `TRUE` (default), write data variables in sub-groups as well.

Returns: Self, invisibly.

`CFLabel` *CF label object*

Description

This class represent CF labels, i.e. a 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 classes

`ncdfCF::CFObject` -> `ncdfCF::CFData` -> `CFLabel`

Active bindings

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

`values` Set or retrieve the labels of this object. In general you should use the `coordinates` field rather than this one. Upon setting values, if there is a linked netCDF resource, this object will be detached from it.

`coordinates` (read-only) Retrieve the labels of this object. Upon retrieval, label values are read from the netCDF resource, if there is one, upon first access and cached thereafter.

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

`dimid` The netCDF dimension id of this label set. Setting this value to anything other than the correct value will lead to disaster.

Methods

Public methods:

- `CFLabel$new()`
- `CFLabel$print()`
- `CFLabel$identical()`
- `CFLabel$copy()`
- `CFLabel$slice()`
- `CFLabel$subset()`
- `CFLabel$write()`

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

Usage:

```
CFLabel$new(var, group, values = NA, start = NA, count = NA)
```

Arguments:

`var` The NCVariable instance upon which this CF object is based when read from a netCDF resource, or the name for the object new CF object to be created.

`group` The CFGroup that this instance will live in.

`values` Optional. The labels of the CF object. Ignored when argument `var` is a `NCVariable` object.

`start` Optional. Integer index value indicating where to start reading data from the file. The value may be `NA` (default) to read all data, otherwise it must not be larger than the number of labels. Ignored when argument `var` is not an `NCVariable` instance.

`count` Optional. Integer value indicating the number of labels to read from file. The value may be `NA` to read to the end of the labels, otherwise its value must agree with the corresponding `start` value and the number of labels on file. Ignored when argument `var` is not an `NCVariable` instance.

Returns: A `CFLabel` instance.

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.

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

Usage:

```
CFLabel$identical(lbl)
```

Arguments:

`lbl` The `CFLabel` instance to test.

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

Method `copy()`**:** Create a copy of this label set. The copy is completely separate from `self`, meaning that both `self` and all of its components are made from new instances.

Usage:

```
CFLabel$copy(name = "", group)
```

Arguments:

`name` The name for the new label set. If an empty string is passed, will use the name of this label set.

`group` The CFGroup where the copy of this axis will live.

Returns: The newly created label set.

Method `slice()`**:** Given a range of domain coordinate values, returns the indices into the axis that fall within the supplied range.

Usage:

```
CFLabel$slice(rng)
```

Arguments:

`rng` A character vector whose extreme (alphabetic) values indicate the indices of coordinates to return.

Returns: An integer vector of length 2 with the lower and higher indices into the axis that fall within the range of coordinates in argument `rng`. Returns `NULL` if no values of the axis fall within the range of coordinates.

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

Usage:

```
CFLabel$subset(name, group, rng)
```

Arguments:

`name` The name for the new label set, optional.

`group` The CFGroup where the copy of this label set will live.

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

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()
```

Returns: Self, invisibly.

`CFObject` *CF base object*

Description

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

Active bindings

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

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

`name` Set or retrieve the name of the CF object. The name must be a valid netCDF name: start with a character, use only characters, numbers and the underscore, and be at most 255 characters long.

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

`group` Set or retrieve the CFGroup that this object is located in, possibly `NULL`.

`attributes` (read-only) Retrieve a `data.frame` with the attributes of the CF object.

`has_resource` (read-only) Flag that indicates if this object has an underlying netCDF resource.

`NC` (read-only) The NC object that links to an underlying netCDF resource, or `NULL` if not linked.

`is_dirty` Flag to indicate if the object has any unsaved changes.

Methods

Public methods:

- CFObject$new()
- CFObject$attach_to_group()
- CFObject$detach()
- CFObject$attribute()
- CFObject$print_attributes()
- CFObject$set_attribute()
- CFObject$attributes_identical()
- CFObject$append_attribute()
- CFObject$delete_attribute()
- CFObject$write_attributes()
- CFObject$clone()

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

Usage:

```
CFObject$new(obj, attributes = data.frame(), group = NULL)
```

Arguments:

`obj` The NCObject instance upon which this CF object is based when read from a netCDF resource, or the name for the new CF object to be created.

`attributes` Optional. A `data.frame` with the attributes of the object. When argument `obj` is an `NCGroup` instance and this argument is an empty `data.frame` (default), arguments will be read from the resource.

`group` The CFGroup instance that this object will live in. The default is `NULL` but this is only useful for `CFGroup` instance.

Returns: A `CFObject` instance.

Method `attach_to_group()`**:** Attach this CF object to a group. If there is another object with the same name in this group an error is thrown. This is the basic method that may be overridden by descendant classes.

Usage:

```
CFObject$attach_to_group(grp, locations = list())
```

Arguments:

`grp` An instance of CFGroup.

`locations` Optional. A `list` whose named elements correspond to the names of objects, possibly including this object. Each list element has a single character string indicating the group in the hierarchy where the object should be stored. As an example, if a data variable has axes "lon" and "lat" and they should be stored in the parent group of `grp`, then specify `locations = list(lon = "..", lat = "..")`. Locations can use absolute paths or relative paths from group `grp`. If the argument `locations` is not provided or the name of the object is not in the list, the object will be stored in group `grp`.

Returns: Self, invisibly.

Method `detach()`**:** Detach the current object from its underlying netCDF resource.

Usage:

```
CFObject$detach()
```

Method `attribute()`**:** Retrieve an attribute of a CF object.

Usage:

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

Arguments:

`att` Single character string of attribute 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 = 30L)
```

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 `attributes_identical()`**:** Test if the supplied attributes are identical to the attributes of this instance. The order of the attributes may differ but the names, types and values must coincide.

Usage:

```
CFObject$attributes_identical(cmp)
```

Arguments:

cmp data.frame with attributes to compare to the attributes of this instance.

Returns: TRUE if attributes in argument cmp are identical to the attributes of this instance, FALSE otherwise.

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 attributes. If an attribute name is not present this method simply returns.

Usage:

```
CFObject$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:

```
CFObject$write_attributes()
```

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.

CFStandardNames *CF Standard names table*

Description

The CF Metadata Conventions define a large number of standard names for physical parameters, including axes and data variables. This class accesses the standard names table. For each of the entries in the table two properties are provided: the canonical unit and a description. These properties are retrieved when searching for a given name.

Access to this class is through the `CF` environment. Use the `CF$standard_names$find("name_of_interest")` method to access a particular standard name. It is strongly recommended not to instantiate this class manually as that may introduce problems with accessing the underlying XML file.

The XML table is retrieved from the CF Metadata Conventions web site here and stored locally in the cache of the `ncdfCF` package. A check is performed periodically for an updated version, which will then be downloaded automatically. The frequency of the update check can be controlled with the `CF.options$cache_stale_days` option.

Active bindings

`is_loaded` (read-only) Flag to determine if the standard names table is available.

Methods

Public methods:

- `CFStandardNames$new()`
- `CFStandardNames$print()`
- `CFStandardNames$find()`
- `CFStandardNames$load()`
- `CFStandardNames$clone()`

Method `new()`**:** Initialize an instance of this class. This is done automatically when the package is loaded.

Usage:

```
CFStandardNames$new()
```

Method `print()`**:** Print the version number of the standard names table in use, if it is loaded. The table is loaded automatically when it is first used.

Usage:

```
CFStandardNames$print()
```

Method `find()`**:** Retrieve the information on the specified names.

Usage:

```
CFStandardNames$find(names)
```

Arguments:

`names` A character vector with the names to search the standard names table for.

Returns: If an entry with a value in `names` is found, returns a `data.frame` with with with the canonical units and a description of the name. If no `names` are found in the table `NULL` is returned.

Method `load()`**:** Load the standard names table so that it's contents may be used in display and analysis. Note that the table may be downloaded (4.3MB at version 91) if not available or stale.

Usage:

```
CFStandardNames$load()
```

Returns: Self, invisibly.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

References

https://cfconventions.org/cf-conventions/cf-conventions.html#standard-name

---

`CFVariable` *CF data variable*

Description

This class represents 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.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFData` -> `CFVariable`

Active bindings

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

`axes` (read-only) List of instances of classes descending from CFAxis that are the axes of the data object.

`ancillary_variables` A list of ancillary data variables associated with this data variable.

`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_measures` (read-only) List of the CFCellMeasure objects of this variable, if defined.

`dimids` (read-only) Retrieve the dimension ids used by the NC variable used by this variable.

`dimnames` (read-only) Retrieve dimnames of the data variable.

`auxiliary_names` (read-only) Retrieve the names of the auxiliary longitude and latitude grids as a vector of two character strings, in that order. If no auxiliary grids are defined, returns `NULL`.

`values` (read-only) Retrieve the raw values of the data variable. In general you should use the `raw()` function rather than this method because the `raw()` function will attach `dimnames` to the array that is returned.

`gridLongLat` Retrieve or set 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$peek()`
- `CFVariable$detach()`
- `CFVariable$time()`
- `CFVariable$raw()`
- `CFVariable$array()`
- `CFVariable$subset()`
- `CFVariable$summarise()`
- `CFVariable$profile()`
- `CFVariable$append()`
- `CFVariable$is_coincident()`
- `CFVariable$add_cell_measure()`
- `CFVariable$add_auxiliary_coordinate()`
- `CFVariable$add_ancillary_variable()`
- `CFVariable$attach_to_group()`
- `CFVariable$terra()`
- `CFVariable$data.table()`
- `CFVariable$write()`
- `CFVariable$save()`

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

Usage:

```
CFVariable$new(
  var,
  group,
```

```
  axes,
  values = values,
  start = NA,
  count = NA,
  attributes = data.frame()
)
```

Arguments:

`var` The NCVariable instance upon which this CF variable is based when read from a netCDF resource, or the name for the new CF variable to be created.

`group` The CFGroup that this instance will live in.

`axes` List of instances of CFAxis to use with this variable.

`values` Optional. The values of the variable in an array.

`start` Optional. Vector of indices where to start reading data along the dimensions of the NC variable on file. The vector must be `NA` to read all data, otherwise it must have agree with the dimensions of the NC variable. Ignored when argument `var` is not an `NCVariable` instance.

`count` Optional. Vector of number of elements to read along each dimension of the NC variable on file. The vector must be `NA` to read to the end of each dimension, otherwise its value must agree with the corresponding `start` value and the dimension of the NC variable. Ignored when argument `var` is not an `NCVariable` instance.

`attributes` Optional. A `data.frame` with the attributes of the object. When argument `var` is an `NCVariable` instance and this argument is an empty `data.frame` (default), arguments will be read from the netCDF resource.

Returns: A `CFVariable` instance.

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()
```

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

Method `detach()`**:** Detach the various properties of this variable from an underlying netCDF resource.

Usage:

```
CFVariable$detach()
```

Returns: Self, invisibly.

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

Usage:

```
CFVariable$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 `raw()`**:** Retrieve the data in the object exactly as it was read from a netCDF resource or produced by an operation.

Usage:

```
CFVariable$raw()
```

Returns: An `array`, `matrix` or `vector` with (dim)names set.

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:

```
CFVariable$array()
```

Returns: An `array` or `matrix` of data in R ordering, or a `vector` if the data has only a single dimension.

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(..., rightmost.closed = FALSE, .resolution = NULL)
```

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.

`rightmost.closed` Single logical value to indicate if the upper boundary of range in each axis should be included. You must use the argument name when specifying this, like `rightmost.closed = TRUE`, to avoid the argument being treated as an axis name.

`.resolution` For interpolation with auxiliary coordinates, the resolution in longitude and latitude directions as numeric values in decimal degrees, optional. If a single value is given, it will apply in both directions. If not supplied, the resolution in the center of the requested area will be calculated and applied over the entire area.

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 `...` extends the valid range of the axis, 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 preserved. The order of the axes in argument `...` does not reorder the axes in the result; use the `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"))`. 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 you can specify their names (instead of specifying the names of the primary planar axes). 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 axes.

Returns: A `CFVariable` instance, having the 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.
If `self` is linked to a netCDF resource then the result will be linked to the same netCDF resource as well, except when *auxiliary coordinate variables* have been selected for the planar axes. In all cases the result will be attached to a private group.

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

Usage:

```
CFVariable$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, quite possibly 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`.

Details: 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 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 `CFVariable$set_attribute()` on the result of this method to set or update attributes as appropriate.

Returns: A `CFVariable` object, or a list thereof with as many `CFVariable` objects as `fun` returns values, or `NULL` if the `era` argument falls entirely outside of the range of the time axis.

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:

```
CFVariable$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 resulting `CFVariable` 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 `CFVariable` 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 `CFVariable` instances with different dimensionality and/or different axes.

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*. If the variable has such *auxiliary coordinate variables* then you can specify their names (instead of specifying the names of the primary planar axes).

Returns: If `.as_table == FALSE`, a `CFVariable` instance, or a list thereof with 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 `append()`**:** Append the data from another `CFVariable` 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. The attributes are not assessed.

Usage:

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

Arguments:

`from` The `CFVariable` instance to append to this data variable.

`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 this data variable and in the `CFVariable` instance in argument `from`.

Returns: Self, invisibly, with the arrays from this data variable and `from` appended, in a new private group.

Method `is_coincident()`**:** Tests if the `other` object is coincident with this data variable: identical axes.

Usage:

```
CFVariable$is_coincident(other)
```

Arguments:

`other` A `CFVariable` instance to compare to this data variable.

Returns: `TRUE` if the data variables are coincident, `FALSE` otherwise.

Method `add_cell_measure()`**:** Add a cell measure variable to this variable.

Usage:

```
CFVariable$add_cell_measure(cm)
```

Arguments:

`cm` An instance of CFCellMeasure.

Returns: Self, invisibly.

Method `add_auxiliary_coordinate()`**:** Add an auxiliary coordinate to the appropriate axis of this variable. The length of the axis must be the same as the length of the auxiliary labels.

Usage:

```
CFVariable$add_auxiliary_coordinate(aux, axis)
```

Arguments:

`aux` An instance of CFLabel or CFAxis.

`axis` An instance of `CFAxis` that these auxiliary coordinates are for.

Returns: Self, invisibly.

Method `add_ancillary_variable()`**:** Add an ancillary variable to this variable.

Usage:

```
CFVariable$add_ancillary_variable(var)
```

Arguments:

`var` An instance of CFVariable.

Returns: Self, invisibly.

Method `attach_to_group()`**:** Attach this variable to a group. If there is another object with the same name in this group an error is thrown. For associated objects (such as axes, CRS, boundary variables, etc), if another object with the same name is otherwise identical to the associated object then that object will be linked from the variable, otherwise an error is thrown.

Usage:

```
CFVariable$attach_to_group(grp, locations = list())
```

Arguments:

`grp` An instance of CFGroup.

`locations` Optional. A `list` whose named elements correspond to the names of objects associated with this variable (but not the variable itself). Each list element has a single character string indicating the group in the hierarchy where the object should be stored. As an example, if the variable has axes "lon" and "lat" and they should be stored in the parent group of `grp`, then specify `locations = list(lon = "..", lat = "..")`. Locations can use absolute paths or relative paths from the group. Associated objects that are not in the list will be stored in group `grp`. If the argument `locations` is not provided, all associated objects will be stored in this group.

Returns: Self, invisibly.

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:

```
CFVariable$terra()
```

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

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

The attributes associated with this data variable will be mostly lost. If present, attributes 'long_name' and 'units' are attached to the `data.table` as attributes, but all others are lost.

Usage:

```
CFVariable$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.tables` are merged by rows 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 `write()`**:** Write the data variable to a netCDF file, including all of its dependent objects, such as axes and attributes.

Axes with `length == 1L` are written as a "scalar axis", unless they are unlimited.

Usage:

```
CFVariable$write(pack)
```

Arguments:

`pack` Optional. Logical to indicate if the data should be packed for a `CFVariable` first written to file. 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. If the variable has been written before, the packing state of the variable on file will be used.

Returns: Self, invisibly.

Method `save()`**:** Save the data variable to a netCDF file, including its subordinate objects such as axes, CRS, etc. Note that saving a data variable will create a "bare-bones" netCDF file and its associated CFDataset.

Usage:

```
CFVariable$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: The newly create `CFDataset`, invisibly.

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::CFData -> ncdfCF::CFVariable -> CFVariableL3b

Methods

Public methods:

- CFVariableL3b$new()
- CFVariableL3b$subset()

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 `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(..., 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.

`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 CFVariable$array() method for this.

Returns: A CFVariable 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.

References

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

---

`CFVerticalParametricTerm`
Parametric formula term for a vertical CF axis object

Description

This class represents a formula term for a parametric vertical axis.

Super classes

`ncdfCF::CFObject` -> `ncdfCF::CFData` -> `ncdfCF::CFVariable` -> `CFVerticalParametricTerm`

Active bindings

`has_data` Logical flag that indicates of the instance has an associated data variable. If not, the instance will report `0` as its data.

`values` (read-only) The values of the parametric term. Depending on the definition of the term, this could be a large array or a simple scalar. Specifically, if the term is defined but no data is included in the netCDF resource, this method will return `0`, as per the CF Metadata Conventions.

Methods

Public methods:

- `CFVerticalParametricTerm$new()`
- `CFVerticalParametricTerm$print()`
- `CFVerticalParametricTerm$subset()`

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

Usage:

```
CFVerticalParametricTerm$new(
  var,
  group,
  axes,
  values = values,
  start = NA,
  count = NA,
  attributes = data.frame()
)
```

Arguments:

`var` The NCVariable instance upon which this CF variable is based when read from a netCDF resource, or the name for the new CF variable to be created.

`group` The CFGroup that this instance will live in.

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

`values` Optional. The values of the variable in an array.

`start` Optional. Vector of indices where to start reading data along the dimensions of the array on file. The vector must be `NA` to read all data, otherwise it must have agree with the dimensions of the array on file. Ignored when argument `var` is not an `NCVariable` instance.

`count` Optional. Vector of number of elements to read along each dimension of the array on file. The vector must be `NA` to read to the end of each dimension, otherwise its value must agree with the corresponding `start` value and the dimension of the array on file. Ignored when argument `var` is not an `NCVariable` instance.

`attributes` Optional. A `data.frame` with the attributes of the object. When argument `var` is an `NCVariable` instance and this argument is an empty `data.frame` (default), arguments will be read from the resource.

Returns: An instance of this class.

Method `print()`**:** Prints a summary of the parametric formula term to the console.

Usage:

```
CFVerticalParametricTerm$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()`**:** Subset the indices to read a smaller portion of the data from the netCDF file. The passed indices should be named after the axes that they refer to. There may be more indices than axes and they may be in a different order than the axes of the term.

Usage:

```
CFVerticalParametricTerm$subset(
  original_axis_names,
  new_axes,
  start,
```

```
  count,
  aux = NULL,
  ZT_dim = NULL
)
```

Arguments:

`original_axis_names` Character vector of names of the axes prior to a modifying operation in the owning data variable.

`new_axes` List of `CFAxis` instances to use for the subsetting.

`start` The indices to start reading data from the file, as an integer vector at least as long as the number of axis for the term.

`count` The number of values to read from the file, as an integer vector at least as long as the number of axis for the term.

`aux` Optional. List with the parameters for an auxiliary grid transformation. Default is `NULL`.

`ZT_dim` Optional. Dimensions of the non-grid axes when an auxiliary grid transformation is specified.

Returns: The new parametric term object.

`create_ncdf` *Create a new data set*

Description

This function creates a new, empty data set in memory. You can add groups, data variables and attributes to this data set as appropriate. The data set can be saved with `CFDataset$save()`, if so desired.

Usage

```
create_ncdf()
```

`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`	A `CFVariable` instance or a descendant of `CFAxis`.

Value

For a `CFVariable` instance in argument `x`, a named vector of axis lengths, excluding any scalar axes. For a `CFAxis` descendant instance in argument `x`, the length of the axis.

Examples

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

geom_ncdf *Create a plot object for a* CFVariable

Description

This is a basic function to support plotting of ncdfCF data with the ggplot2 package. Specifically, this function creates a geo_ncdf object which can be used like a geom_raster. The geom_ncdf takes a CFVariable instance as its data. The CFVariable should be properly pre-processed to make it suitable for plotting. The $subset() method is well suited for this task. Note that currently only map plotting works, e.g. the CFVariable should have X and Y axes.

Usage

```
geom_ncdf(mapping = NULL, data, ...)
```

Arguments

mapping	As in geom_raster. If the argument is not provided, a mapping is constructed from the properties of the data argument, which is usually the right way.
data	A CFVariable instance. This will override any data setting of the ggplot() function.
...	Arguments passed on to geom_raster().

Value

A geom_* object that can be used in ggplot2 plot composition.

Examples

```
library(ggplot2)
fn <- system.file("extdata", "tasmax_NAM-44_day_20410701-vncdfCF.nc", package = "ncdfCF")
ds <- open_ncdf(fn)
tasmax <- ds[["tasmax"]]
ggplot() + geom_ncdf(data = tasmax) + coord_equal() + scale_fill_viridis_c()
```

groups *List the groups in the CF object, recursively.*

Description

List the groups in the CF object, recursively.

Usage

```
groups(x)

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

Arguments

x	A CFDataset instance.

Value

A character vector with group names in the object.

Examples

```
fn <- system.file("extdata", "ERA5land_Rwanda_20160101.nc", package = "ncdfCF")
ds <- open_ncdf(fn)
groups(ds)
```

---

makeAxis *Create an axis*

Description

With this function you can create an axis to use with new CFVariable 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,
  orientation,
  values,
  bounds = NULL,
  attributes = data.frame(),
  group = NULL
)
```

Arguments

name	Name of the axis.
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. In the case of an axis with orientation = "T" this must be a CFTime or CFClimatology instance.
bounds	The boundary values of the coordinates, or NULL if not available.
attributes	data.frame with the attributes of the axis to create. Depending on which axis is created one or more attributes may be added or amended.
group	CFGroup instance where the axis will be located. If NULL (default), a private group will be created for the axis.

Details

There are several restrictions on the combination of orientation and values arguments. Longitude, latitude and depth axes (orientation of "X", "Y" or "Z") 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()

---

makeCharacterAxis *Create a character axis*

Description

With this method you can create a character axis to use with new CFVariable instances.

Usage

```
makeCharacterAxis(name, values, attributes = data.frame(), group = NULL)
```

Arguments

name	Name of the axis.
values	The character coordinate values of the axis.
attributes	data.frame with the attributes of the axis to create.
group	CFGroup instance where the axis will be located. If NULL (default), a private group will be created for the axis.

Value

A CFAxisCharacter instance.

makeDiscreteAxis *Create a discrete axis*

Description

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

Usage

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

Arguments

name	Name of the axis.
length	The length of the axis.
group	CFGroup instance where the axis will be located. If NULL (default), a private group will be created for the axis.

Value

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

makeGroup *Create a new group*

Description

This function creates a new group for CF objects. Groups are organized in an hierarchy, starting with a root node. A root node is specified by an empty string for a name and with argument parent_group = NULL.

Usage

```
makeGroup(name = "", parent_group = NULL)
```

Arguments

name	A name for the group. This must be a valid name for CF objects. The default value is the empty string "".
parent_group	Optionally, a parent for the current group. This must be an instance of CFGroup itself.

Value

A new instance of CFGroup.

Examples

```
root <- makeGroup()
sub_group <- makeGroup("sub", root)
```

`makeLatitudeAxis` *Create a latitude axis*

Description

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

Usage

```
makeLatitudeAxis(
  name,
  values,
  bounds = NULL,
  attributes = data.frame(),
  group = NULL
)
```

Arguments

`name`	Name of the axis.
`values`	The coordinate values.
`bounds`	The bounds of the coordinate values, or `NULL` if not available.
`attributes`	`data.frame` with the attributes of the axis to create. Attributes "standard_name", "units", "actual_range" and "axis" will be set or updated.
`group`	CFGroup instance where the axis will be located. If `NULL` (default), a private group will be created for the axis.

Value

A CFAxisLatitude instance.

makeLongitudeAxis *Create a longitude axis*

Description

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

Usage

```
makeLongitudeAxis(
  name,
  values,
  bounds = NULL,
  attributes = data.frame(),
  group = NULL
)
```

Arguments

name	Name of the axis.
values	The coordinate values.
bounds	The bounds of the coordinate values, or NULL if not available.
attributes	data.frame with the attributes of the axis to create. Attributes "standard_name", "units", "actual_range" and "axis" will be set or updated.
group	CFGroup instance where the axis will be located. If NULL (default), a private group will be created for the axis.

Value

A CFAxisLongitude instance.

makeTimeAxis *Create a time axis*

Description

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

Usage

```
makeTimeAxis(name, values, attributes = data.frame(), group = NULL)
```

Arguments

name	Name of the axis.
values	A CFTime or CFClimatology instance with time values and optionally bounds set.
attributes	data.frame with the attributes of the axis to create. Attributes "standard_name", "units", "calendar", "actual_range" and "axis" will be set or updated.
group	CFGroup instance where the axis will be located. If NULL (default), a private group will be created for the axis.

Value

A CFAxisTime instance.

---

makeVerticalAxis *Create a vertical axis*

Description

With this method you can create a vertical axis to use with new CFVariable instances. Note that you should set the "positive" attribute after creating the axis to indicate if values are increasing going upwards (positive = "up") or downwards (positive = "down").

Usage

```
makeVerticalAxis(
  name,
  values,
  bounds = NULL,
  attributes = data.frame(),
  group = NULL
)
```

Arguments

name	Name of the axis.
values	The coordinate values.
bounds	The bounds of the coordinate values, or NULL if not available.
attributes	data.frame with the attributes of the axis to create. Attributes "actual_range" and "axis" will be set or updated.
group	CFGroup instance where the axis will be located. If NULL (default), a private group will be created for the axis.

Value

A CFAxisVertical instance.

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

Description

Retrieve the variable or axis names of an `ncdfCF` object. The `names()` function gives the names of the variables in the data set, preceded by 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, preceded 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 coordinate values along the axis as a numeric vector.
- `CFAxisTime`: The coordinate values 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 coordinate values along the axis as a character vector.
- `CFAxisDiscrete`: The index values of the axis, either along the entire axis, or a portion thereof.

Usage

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

# Names of data variables
names(ds)

# 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 dimension. It contains the information on a dimension that is stored in an netCDF file. Consequently, the properties of this class are all read-only. The length of the dimension may change if data is written to an unlimited dimension, but that is managed internally.

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

Super class

`ncdfCF::NCObject` -> `NCDimension`

Active bindings

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

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

Methods

Public methods:

- `NCDimension$new()`
- `NCDimension$print()`
- `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 = 1L, unlim = FALSE, group)
```

Arguments:

`id` Numeric identifier of the netCDF dimension.

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

`length` Length of the dimension. Default is 1.

`unlim` Is the dimension unlimited? Default is `FALSE`.

`group` The NC group where the dimension is located.

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

Usage:

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

Arguments:

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

Returns: Self, invisibly.

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.

Direct access to groups is usually not necessary. The principal objects of interest, CF data variables and axes, are accessible via CFGroup. Changing the properties of a netCDF group other than its name may very well invalidate the CF objects or even the netCDF file.

Super class

`ncdfCF::NCObject` -> `NCGroup`

Public fields

`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.

Active bindings

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

`resource` (read-only) The RNetCDF object to the underlying netCDF resource.

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

`can_write` (read-only) Is the resource writable?

`name` Set or retrieve the name of the group. Note that the name is always relative to the location in the hierarchy that the group is in and it should thus not be qualified by backslashes. The name has to be a valid CF name. The name of the root group cannot be changed.

`fullname` (read-only) The fully qualified absolute path of the group.

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

`CF` Set or retrieve the CFGroup that is associated with this NC group.

Methods

Public methods:

- `NCGroup$new()`
- `NCGroup$print()`
- `NCGroup$find_by_name()`
- `NCGroup$find_dim_by_id()`
- `NCGroup$has_name()`
- `NCGroup$set_name()`
- `NCGroup$unused()`
- `NCGroup$create_group()`
- `NCGroup$append()`
- `NCGroup$fullnames()`
- `NCGroup$dimensions()`
- `NCGroup$clone()`

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

Usage:

```
NCGroup$new(id, name, attributes = data.frame(), parent, resource)
```

Arguments:

`id` The identifier of the group. If `NA`, the new group will be created in the netCDF resource, unless argument `parent == NULL`, i.e. the root group which already exists.

`name` The name of the group.

`attributes` Optional, a `data.frame` with group attributes.

`parent` The parent group of this group. If `NULL` then argument `resource` must be a valid instance of `NCResource`.

`resource` Optional. Reference to the NCResource instance that provides access to the netCDF resource.

Returns: An instance of this class.

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 `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. Usually this method is called programmatically.

Usage:

```
NCGroup$find_by_name(name)
```

Arguments:

`name` The name of an object, with an optional absolute or relative group path from the calling group. The object must be an NC group, dimension or variable.

Returns: The object with the provided name. 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)
```

Arguments:

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

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

Method `set_name()`**:** Change the name of the NC group. The new name must be valid and should not duplicate a sibling group.

Usage:

```
NCGroup$set_name(new_name)
```

Arguments:

`new_name` The new name for the NC group.

Returns: Self, invisibly.

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

Usage:

```
NCGroup$unused()
```

Returns: List of NCVariable.

Method `create_group()`**:** Create a new group as a sub-group of the current group. This writes the new group to the netCDF resource, but only if it is open for writing.

Usage:

```
NCGroup$create_group(CFgroup)
```

Arguments:

`CFgroup` The CFGroup associated with this NC group.

Returns: The newly created group as a `NCGroup` instance, invisibly.

Method `append()`**:** Append an object to this group.

Usage:

```
NCGroup$append(obj)
```

Arguments:

`obj` The object to append. This must be an `NCVariable` or `NCDimension` instance. Any other type of object will generate a warning.

Returns: Self, invisible.

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, possibly 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(scope = "all")
```

Arguments:

`scope` Character string that indicates if only dimensions in the current group should be reported (`local`) or visible dimensions in parent groups as well (`all`, default).

Returns: A vector of NCDimension objects.

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.

Active bindings

`id` (read-only) Retrieve the identifier of the netCDF object.

`name` Set or retrieve the name of the NC object. The netCDF file must be open for writing to change the name.

`attributes` (read-only) Read the attributes of the object. When there are no attributes, an empty `data.frame` will be returned.

`CF` Register CF object that uses this netCDF object, or retrieve the list of registered CF objects.

Methods

Public methods:

- `NCObject$new()`
- `NCObject$print_attributes()`
- `NCObject$attribute()`
- `NCObject$write_attributes()`
- `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, attributes = data.frame())
```

Arguments:

`id` Numeric identifier of the netCDF object.

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

`attributes` Optional, `data.frame` with attributes of the object.

Method `print_attributes()`**:** This function prints the attributes of the netCDF object to the console.

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()`**:** Retrieve an attribute of a NC object.

Usage:

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

Arguments:

`att` Single character string of attribute 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 `write_attributes()`**:** Write the attributes of this object to a netCDF file. This will retain existing attributes, update modified attributes, and delete and add missing attributes from the passed in argument.

Usage:

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

Arguments:

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

`new_atts` The attributes to write.

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.

NCResource *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 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.

`can_write` (read-only) Is the resource writable?

Methods

Public methods:

- `NCResource$new()`
- `NCResource$print()`
- `NCResource$create()`
- `NCResource$close()`
- `NCResource$group_handle()`
- `NCResource$clone()`

Method `new()`**:** Create a connection to a netCDF resource. This is called by `open_ncdf()` when opening a netCDF resource or when saving a dataset to file. You should never have to call this directly.

Usage:

```
NCResource$new(uri, write)
```

Arguments:

`uri` The URI to the netCDF resource.

`write` Logical flag to indicate if the resource should be read-write.

Returns: An instance of this class.

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

Usage:

```
NCResource$print()
```

Returns: Self, invisibly.

Method `create()`**:** Create a new file on disk for the netCDF resource.

Usage:

```
NCResource$create()
```

Returns: Self, invisibly.

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

Usage:

```
NCResource$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:

```
NCResource$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:

```
NCResource$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`

Active bindings

`group` (read-only) NetCDF group where this variable is located.

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

`vtype` (read-only) 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` (read-only) Number of dimensions that this variable uses.

`dimids` (read-only) Vector of dimension identifiers that this NCVariable uses.

`netcdf4` (read-only) Additional properties for a `netcdf4` resource.

`CF` Register CF objects that use this netCDF variable, or retrieve the list of registered CF objects.

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

`is_packed` (read-only) Flag that indicates if the data on file is packed.

Methods

Public methods:

- `NCVariable$new()`
- `NCVariable$print()`
- `NCVariable$shard()`
- `NCVariable$detach()`
- `NCVariable$get_data()`
- `NCVariable$write_data()`
- `NCVariable$set_name()`
- `NCVariable$dimension()`
- `NCVariable$dim()`
- `NCVariable$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,
  dimids,
  attributes = data.frame(),
  netcdf4 = list()
)
```

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.

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

`attributes` Optional, `data.frame` with the attributes of the object.

`netcdf4` Optional, `netcdf4`-specific arguments in the format of RNetCDF.

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 `detach()`**:** Detach the passed object from this NC variable.

Usage:

```
NCVariable$detach(obj)
```

Arguments:

`obj` The CFObject instance to detach from this NC variable.

Returns: `obj`, invisibly.

Method `get_data()`**:** Read (a chunk of) data from the netCDF file. Degenerate dimensions are maintained and data is always returned in its smallest type.

Usage:

```
NCVariable$get_data(start = NA, count = NA)
```

Arguments:

`start` A vector of indices specifying the element where reading starts along each dimension of the data. When `NA`, all data are read from the start of the array.

`count` An integer vector specifying the number of values to read along each dimension of the data. Any `NA` value in vector count indicates that the corresponding dimension should be read from the start index to the end of the dimension.

Returns: An array, matrix or vector with the requested data, or an error object.

Method `write_data()`**:** Write (a chunk of) data to the netCDF file.

Usage:

```
NCVariable$write_data(d, start = NA, count = NA, ...)
```

Arguments:

`d` The data to write. This must have appropriate dimensions.

`start` A vector of indices specifying the element where writing starts along each dimension of the data. When `NA`, all data are written from the start of the array.

`count` An integer vector specifying the number of values to write along each dimension of the data. Any `NA` value in vector count indicates that the corresponding dimension should be written from the start index to the end of the dimension.

`...` Other parameters passed on to `RNetCDF::var.put.nc()`.

Returns: Self, invisibly.

Method `set_name()`**:** Change the name of the NC variable. The new name must be valid in the indicated group, it can not already exist in the group. The netCDF file must be open for writing to change the name.

Usage:

```
NCVariable$set_name(new_name)
```

Arguments:

`new_name` The new name for the NC variable

Returns: Self, invisibly.

Method `dimension()`**:** Get the NCDimension object(s) that this variable uses.

Usage:

```
NCVariable$dimension(id)
```

Arguments:

`id` The index of the dimension. If missing, all dimensions of this variable are returned.

Returns: A `NCDimension` object or a list thereof. If no `NCDimensions` were found, return `NULL`.

Method `dim()`**:** The lengths of the data dimensions of this object.

Usage:

```
NCVariable$dim(dimension)
```

Arguments:

`dimension` Optional. The index of the dimension to retrieve the length for. If omitted, retrieve the lengths of all dimensions.

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, write = FALSE)
```

Arguments

resource	The name of the netCDF resource to open, either a local file name or a remote URI.
write	TRUE if the file is to be opened for writing, FALSE (default) for read-only access. Ignored for online resources, which are always opened for read-only access.

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))
```

Ops.CFVariable *Operations on CFVariable objects*

Description

Basic arithmetic, mathematical and logical operations can be applied on the data of CFVariable objects having a suitable data type, specifically the base R functions from the Ops and Math groups of the S3 groupGeneric functions.

Usage

```
## S3 method for class 'CFVariable'
Ops(e1, e2)

## S3 method for class 'CFVariable'
Math(x, ...)
```

Arguments

e1, e2	CFVariable objects, or a single numeric value.
x	A CFVariable object.
...	Additional arguments passed on to the math functions.

Details

The functions always return a new CFVariable object. Functions can thus be concatenated to create more complex expressions. The data type of the new object is determined by the base R function; its name is concatenated from the names in the argument object(s). The result will be assigned to a private group and is thus completely disjoint from other CF objects.

For the Ops functions with two arguments, if both arguments are a CFVariable object they have to be compatible: same shape, axis coordinate values and coordinate reference system. The resulting CFVariable object will use the same axes as the CFVariable object(s) used as argument.

The attributes of the resulting CFVariable object should be updated to reflect its contents, in particular the "name", "long_name", "standard_name" and "units" attributes. Attributes are not copied over from the CFVariable objects in the arguments.

Value

A new CFVariable object. The object will have the same coordinate space as the CFVariable object used as argument. Arguments are not copied and the new object will only have the "actual_range" attribute set.

Results that are logical (see the examples) are stored using the NC_SHORT data type because netCDF does not have a native logical data type.

Examples

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

# Temperature data in K
t2m <- ds[["t2m"]]

# Convert to degrees_Celsius
t2mC <- t2m - 273.15
t2mC$name <- "t2m_Celsius"
t2mC$set_attribute("units", "NC_CHAR", "degrees_Celsius")
t2mC

hot <- t2mC > 20
hot$name <- "t2m_Celsius_over_20"
hot$set_attribute("long_name", "NC_CHAR", "Flag to indicate where temperature is 20C or hotter")
hot$set_attribute("units", "NC_CHAR", "1")
hot
```

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.

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)
```

Index