Package: tidync (via r-universe)

October 29, 2026

Title Tidy Approach to 'NetCDF' Data Exploration and Extraction

Version 0.5.0

Description Tidy tools for 'NetCDF' data sources. Explore the contents of a 'NetCDF' source (file or URL) presented as variables organized by grid with a database-like interface. The hyper_filter() interactive function translates the filter value or index expressions to array-slicing form. No data is read until explicitly requested, as a data frame or list of arrays via hyper_tibble() or hyper_array().

Depends R (>= 4.1.0)

License GPL-3

Encoding UTF-8

ByteCompile true

Imports dplyr (>= 1.1.0), ncdf4, ncmeta (>= 0.5.0), RNetCDF (>= 1.9-1), rlang, tibble, tidyr (>= 1.0.0), CFtime (>= 1.4.0)

Suggests ggplot2, knitr, mirai, rmarkdown, testthat, covr, cubelyr

SystemRequirements netcdf udunits-2

Roxygen list(markdown = TRUE)

URL https://docs.ropensci.org/tidync/

BugReports https://github.com/ropensci/tidync/issues

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

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

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

Date/Publication 2026-07-31 10:57:16 UTC

RemoteUrl https://github.com/ropensci/tidync

RemoteRef HEAD

RemoteSha d503ad4f248793884e51d7888853d286a54a5a18

Contents

activate 2
hyper_array 4
hyper_filter 6
hyper_tbl_cube 7
hyper_tibble 8
hyper_transforms 10
hyper_vars 11
nc_get 12
print.tidync 12
print.tidync_data 14
tidync 15

Index **18**

`activate` *Activate a NetCDF grid*

Description

A grid in NetCDF is a particular shape and size available for array variables, and consists of sets of dimensions. To activate a grid is to set the context for downstream operations, for querying, summarizing and reading data. There's no sense in performing these operations on more than one grid at a time, but multiple variables may exist in a single grid. There may be only one significant grid in a source or many, individual dimensions are themselves grids.

Usage

```
activate(.data, what, ..., select_var = NULL)

## S3 method for class 'tidync'
activate(.data, what, ..., select_var = NULL)

## S3 method for class 'tidync'
active(x)

active(x)

## Default S3 method:
active(x)

active(x) <- value

## Default S3 replacement method:
active(x) <- value
```

Arguments

.data	NetCDF object
what	name of a grid or variable
...	reserved, currently ignored
select_var	optional argument to set selected state of variable/s by name
x	NetCDF object
value	name of grid or variable to be active

Details

There may be more than one grid and one is always activated by default. A grid may be activated by name in the form of 'D1,D0' where one or more numbered dimensions indicates the grid. The grid definition names are printed as part of the summary of in the tidync object and may be obtained directly with hyper_grids() on the tidync object.

Activation of a grid sets the context for downstream operations (slicing and reading data) from NetCDF, and as there may be several grids in a single source activation allows a different choice of available variables. By default the largest grid is activated. Once activated, all downstream tasks apply to the set of variables that exist on that grid.

If activate() is called with a variable name, it puts the variable first. The function active() gets and sets the active grid. To restrict ultimate read to particular variables use the select_var argument to hyper_filter(), hyper_tibble() and hyper_tbl_cube().

Scalar variables are not currently available to tidync, and it's not obvious how activation would occur for scalars, but in future perhaps activate("S") could be the right way forward.

Value

NetCDF object

See Also

hyper_filter hyper_tibble hyper_tbl_cube

Examples

```
if (!tolower(Sys.info()[["sysname"]]) == "sunos") {
 l3file <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
 rnc <- tidync(system.file("extdata", "oceandata", l3file,
 package = "tidync"))
 activate(rnc, "palette")

 ## extract available grid names
 hyper_grids(rnc)
}
```

hyper_array *Extract NetCDF data as an array*

Description

Extract the raw array data as a list of one or more arrays. This can be the entire variable/s or after dimension-slicing using `hyper_filter()` expressions. This is a delay-breaking function and causes data to be read from the source into R native arrays. This list of arrays is lightly classed as tidync_data, with methods for `print()` and `tidync()`.

Usage

```
hyper_array(
  x,
  select_var = NULL,
  ...,
  raw_datavals = FALSE,
  force = FALSE,
  drop = TRUE
)

hyper_slice(
  x,
  select_var = NULL,
  ...,
  raw_datavals = FALSE,
  force = FALSE,
  drop = TRUE
)

## S3 method for class 'tidync'
hyper_array(
  x,
  select_var = NULL,
  ...,
  raw_datavals = FALSE,
  force = FALSE,
  drop = TRUE
)

## S3 method for class 'character'
hyper_array(
  x,
  select_var = NULL,
  ...,
  raw_datavals = FALSE,
  force = FALSE,
```

```
  drop = TRUE
)
```

Arguments

x	NetCDF file, connection object, or tidync object
select_var	optional vector of variable names to select
...	passed to hyper_filter()
raw_datavals	logical to control whether scaling in the NetCDF is applied or not
force	ignore caveats about large extraction and just do it
drop	collapse degenerate dimensions, defaults to TRUE

Details

The function hyper_array() is used by hyper_tibble() and hyper_tbl_cube() to actually extract data arrays from NetCDF, if a result would be particularly large there is a check made and user-opportunity to cancel. This is controllable as an option getOption('tidync.large.data.check'), and can be set to never check with options(tidync.large.data.check = FALSE).

The function hyper_array() will act on an existing tidync object or a source string.

By default all variables in the active grid are returned, use select_var to specify one or more desired variables.

The transforms are stored as a list of tables in an attribute transforms, access these with hyper_transforms().

Value

a named list of arrays classed as tidync_data, one element per variable, with dimension coordinates stored in the transforms attribute

See Also

print.tidync_data for a description of the print summary, hyper_tbl_cube() and hyper_tibble() which are also delay-breaking functions that cause data to be read

Examples

```
f <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
l3file <- system.file("extdata/oceandata", f, package= "tidync")

## extract a raw list by filtered dimension
library(dplyr)
araw1 <- tidync(l3file) |>
 hyper_filter(lat = between(lat, -78, -75.8),
              lon = between(lon, 165, 171)) |>
 hyper_array()

araw <- tidync(l3file) |>
         hyper_filter(lat = abs(lat) < 10,
                      lon = index < 100) |>
```

```
  hyper_array()

## hyper_array will pass the expressions to hyper_filter
braw <- tidync(l3file) |>
  hyper_array(lat = abs(lat) < 10, lon = index < 100)

## get the transforms tables (the axis coordinates)
lapply(attr(braw, "transforms"),
   function(x) nrow(dplyr::filter(x, selected)))
## the selected axis coordinates should match in order and in size
lapply(braw, dim)
```

`hyper_filter` *Subset NetCDF variable by expression*

Description

The `hyper_filter()` acts on a tidync object by matching one or more filtering expressions like with `dplyr::filter`. This allows us to lazily specify a subset from a NetCDF array without pulling any data. The modified object may be printed to see the effects of subsetting, or saved for further use.

Usage

```
hyper_filter(.x, ...)

## S3 method for class 'tidync'
hyper_filter(.x, ...)
```

Arguments

`.x`	NetCDF file, connection object, or `tidync` object
`...`	currently ignored

Details

The function `hyper_filter()` will act on an existing tidync object or a source string.

Filter arguments must be named as per the dimensions in the variable in form `dimname = dimname < 10`. This is a restrictive variant of `dplyr::filter()`, with a syntax more like `dplyr::mutate()`. This ensures that each element is named, so we know which dimension to apply this to, but also that the expression evaluated against can do some extra work for a nuanced test.

There are special columns provided with each axis, one is 'index' so that exact matching can be done by position, or to ignore the actual value of the coordinate. That means we can use a form like `dimname = index < 10` to subset by position in the array index, without necessarily knowing the values along that dimension.

Value

data frame

Examples

```
f <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
l3file <- system.file("extdata/oceandata", f, package= "tidync")
## filter by value
tidync(l3file) |> hyper_filter(lon = lon < 100)
## filter by index
tidync(l3file) |> hyper_filter(lon = index < 100)

## be careful that multiple comparisons must occur in one expression
 tidync(l3file) |> hyper_filter(lon = lon < 100 & lon > 50)

## filter in combination/s
tidync(l3file) |> hyper_filter(lat = abs(lat) < 10, lon = index < 100)
```

---

hyper_tbl_cube *A dplyr cube tbl*

Description

Produce a tbl_cube from NetCDF. This is a delay-breaking function and causes data to be read from the source into the tbl cube format defined in the dplyr package.

Usage

```
hyper_tbl_cube(x, ..., force = FALSE)

## S3 method for class 'tidync'
hyper_tbl_cube(x, ..., force = FALSE)

## S3 method for class 'character'
hyper_tbl_cube(x, ..., force = FALSE)
```

Arguments

x	tidync object
...	arguments for hyper_filter()
force	ignore caveats about large extraction and just do it

Details

The size of an extraction is checked and if *quite large* there is an a user-controlled prompt to proceed or cancel. This can be disabled with `options(tidync.large.data.check = FALSE)`

- please see hyper_array() for more details.

The tbl cube is a very general and arbitrarily-sized array that can be used with tidyverse functionality. Dimension coordinates are stored with the tbl cube, derived from the grid transforms.

Value

tbl_cube

dplyr::tbl_cube

See Also

hyper_array() and hyper_tibble() which are also delay-breaking functions that cause data to be read

Examples

```
f <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
l3file <- system.file("extdata/oceandata", f, package= "tidync")
(cube <- hyper_tbl_cube(tidync(l3file) |>
activate(chlor_a), lon = lon > 107, lat = abs(lat) < 30))
ufile <- system.file("extdata", "unidata", "test_hgroups.nc",
 package = "tidync", mustWork = TRUE)

## some versions of NetCDF don't support this file
## (4.1.3 tidync/issues/82)
group_nc <- try(tidync(ufile), silent = TRUE)
if (!inherits(group_nc, "try-error")) {
 res <-  hyper_tbl_cube(tidync(ufile))
 print(res)
} else {
 ## the error was
 writeLines(c(group_nc))
}
```

---

hyper_tibble *Extract NetCDF data as an expanded table.*

Description

Extract the raw array data as an expanded data frame. This can be the entire variable/s or after dimension-slicing using hyper_filter() expressions with dimension values expanded appropriately for each element in the arrays (one row per element).

Usage

```
hyper_tibble(x, ..., na.rm = TRUE, force = FALSE)

## S3 method for class 'character'
hyper_tibble(x, ..., na.rm = TRUE, force = FALSE)

## S3 method for class 'tidync'
hyper_tibble(x, ..., na.rm = TRUE, force = FALSE)
```

Arguments

x	NetCDF file, connection object, or tidync object
...	arguments to 'hyper_filter"
na.rm	if TRUE these rows are not included in the output when all variables are NA
force	ignore caveats about large extraction and just do it

Details

The size of an extraction is checked and if *quite large* there is an a user-controlled prompt to proceed or cancel. This can be disabled with `options(tidync.large.data.check = FALSE)`

- please see `hyper_array()` for more details.

The function `hyper_tibble()` will act on an existing tidync object or a source string.

By default all variables in the active grid are returned, use `select_var` to limit.

Value

a `tbl_df`

See Also

`hyper_array()` and `hyper_tbl_cube()` which are also delay-breaking functions that cause data to be read

Examples

```
l3file <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
f <- system.file("extdata", "oceandata", l3file, package= "tidync")
rnc <- tidync(f)
hyper_filter(rnc)
library(dplyr)
lapply(hyper_array(f, lat = lat > 0, lon = index > 3000), dim)

ht <- hyper_tibble(rnc) |>
filter(!is.na(chlor_a))
ht
if (requireNamespace("ggplot2")) {
 library(ggplot2)
 ggplot(ht |> filter(!is.na(chlor_a)),
 aes(x = lon, y = lat, fill = chlor_a)) + geom_tile()
}
```

hyper_transforms *Axis transforms*

Description

Axis 'transforms' are data frames of each dimension in a NetCDF source. `hyper_transforms` returns a list of the active transforms by default,

Usage

```
hyper_transforms(x, all = FALSE, ...)

## Default S3 method:
hyper_transforms(x, all = FALSE, ...)
```

Arguments

`x`	tidync object
`all`	set to `TRUE` to return all transforms, not only active ones
`...`	ignored

Details

Each transform is available by name from a list, and each data frame has the coordinate of the dimension, its index, and a 'selected' variable set by the filtering expressions in `hyper_filter` and used by the read-functions `hyper_array` and `hyper_tibble`.

Use `hyper_transforms` to interrogate and explore the available dimension manually, or for development of custom functions.

Value

list of axis transforms

Examples

```
l3file <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
f <- system.file("extdata", "oceandata", l3file, package = "tidync")
ax <- tidync(f) |> hyper_transforms()
names(ax)
lapply(ax, dim)

## this function returns the transforms tidync knows about for this source
str(tidync(f)$transforms)
names(hyper_transforms(tidync(f), all = TRUE))
```

hyper_vars *Grid status*

Description

Functions to report on the current status of the `active` grid. Information on the active dimensions and variables are listed in a data frame with multiple columns.

Usage

```
hyper_vars(x, ...)

hyper_dims(x, ...)

hyper_grids(x, ...)
```

Arguments

x	tidync object
...	ignored

Details

The dimensions and variables of the active grid are identified in the print method of the tidync object, these functions exist to provide that information directly.

`hyper_vars()` will list the ids, data type, name, dimension number, number of attributes and and coordinate status of the variables on the currently active grid.

`hyper_dims()` will list the names, lengths, start/count index, ids, and status of dimensions on the currently active grid. records on the currently active dimensions.

`hyper_grids()` will list the names, number of dimension, and number of variables and active status of each grid in the source.

Value

data frame

Examples

```
f <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
l3file <- system.file("extdata/oceandata", f, package= "tidync")
tnc <- tidync(l3file)
hyper_vars(tnc)
hyper_dims(tnc)
hyper_dims(tnc |> hyper_filter(lat = lat < 20))
```

nc_get *Helper to get a variable from NetCDF.*

Description

This exists so we can (internally) use a file path, uri, or open NetCDF connection (ncdf4 or RNetCDF) in a simpler way.

Usage

```
nc_get(x, v, test = FALSE)
```

Arguments

x	file path, uri, or NetCDF connection
v	variable name
test	if true we make sure the connection can be open, not applied for connections themselves

Details

This function just reads the whole array. It is equivalent to the angstroms package function 'rawdata(x, varname)'.

Value

array of values for the requested variable

Examples

```
l3file <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
fpath <- system.file("extdata", "oceandata", l3file,
package = "tidync")
lat <- nc_get(fpath, "lat")
```

print.tidync *Print tidync object*

Description

Provide a summary of variables and dimensions, organized by their 'grid' (or 'shape') and with a summary of any slicing operations provided as 'start' and 'count' summaries for each dimension in the active grid.

Usage

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

Arguments

x	NetCDF object
...	reserved

Details

See tidync for detail about the object, and hyper_vars for programmatic access to the active grid's variable and dimension information.

The print summary is organized in two sections, the first is available grids (sets of dimensions) and their associated variables, the second is the dimensions, separated into active and inactive. All dimensions may be active in some NetCDF sources.

Individual *active* dimensions include the following components: * 'dim' - dimension label, D0, D1, D2, ... * 'name' - dimension name * 'length' - size of the dimension * 'min' - minimum value of the dimension * 'max' - maximum value of the dimension * 'start' - start index of subsetting * 'count' - length of subsetting index * 'dmin' - minimum value of the subset dimension * 'dmax' - maximum value of the subset dimension * 'unlim'

- indicates whether dimension is unlimited (spread across other files, usually the time-step) * 'coord_dim' - indicates whether dimension is a coordinate-dimension (i.e. listed as a 1-D grid)

The *inactive* dimension summary does not include 'start', 'count', 'dmin', 'dmax' as these are identical to the values of 1, 'length', 'min', 'max' when no array subsetting has been applied.

Value

the input object invisibly

Examples

```
argofile <- system.file("extdata/argo/MD5903593_001.nc", package = "tidync")
argo <- tidync(argofile)
print(argo)

## the print is modified by choosing a new grid or running filters
argo |> activate("D7,D9,D11,D8")

argo |> hyper_filter(N_LEVELS = index > 300)
```

`print.tidync_data` *Print tidync data*

Description

Print method for the 'tidync_data' list of arrays returned by `hyper_array()`.

Usage

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

Arguments

x	'tidync_data' object (from `hyper_array()`)
...	reserved args

Details

The output lists the variables and their dimensions of an object from a previous call to `tidync()`, and possibly `hyper_filter()`. The available data will differ from the source in terms of variables (via `select_var` in hyper_array) and the lengths of each dimension (via named expressions in `hyper_filter()`).

Value

the input object invisibly

See Also

tidync_data

Examples

```
argofile <- system.file("extdata/argo/MD5903593_001.nc", package = "tidync")
argodata <- tidync(argofile) |> hyper_filter(N_LEVELS = index < 5) |>
              hyper_array(select_var = c("TEMP_ADJUSTED", "PRES"))
print(argodata)
```

tidync *Tidy NetCDF*

Description

Connect to a NetCDF source and allow use of `hyper_*` verbs for slicing with `hyper_filter()`, extracting data with `hyper_array()` and [hyper_tibble() from an activated grid. By default the largest *grid* encountered is activated, see`activate()`.

Usage

```
tidync(x, what, ...)

## S3 method for class 'character'
tidync(x, what, ..., concat_dim = NULL, fast = FALSE)

## S3 method for class 'tidync_data'
tidync(x, what, ...)
```

Arguments

`x`	path to a NetCDF file, or a character vector of paths for multi-source access (requires `concat_dim`)
`what`	(optional) character name of grid (see `ncmeta::nc_grids`) or (bare) name of variable (see `ncmeta::nc_vars`) or index of grid to `activate`
`...`	reserved for arguments to methods, currently ignored
`concat_dim`	(optional) name of the dimension to concatenate across sources, or a list with elements `name` (dimension name) and `values` (vector of coordinate values, one per source). The list form avoids opening files 2..N entirely — useful when values are already known from a file database. Values can be numeric, Date, or POSIXct.
`fast`	logical, if `TRUE` skip full metadata validation for sources after the first (only reads the `concat_dim` coordinate). Mismatches are detected at data-read time. Ignored when `concat_dim$values` is supplied.

Details

The print method for tidync includes a lot of information about which variables exist on which dimensions, and if any slicing (`hyper_filter()`) operations have occurred these are summarized as 'start' and 'count' modifications relative to the dimension lengths. See print for these details, and hyper_vars for programmatic access to this information

Many NetCDF forms are supported and tidync tries to reduce the interpretation applied to a given source. The NetCDF system defines a 'grid' for storing array data, where 'grid' is the array 'shape', or 'set of dimensions'). There may be several grids in a single source and so we introduce the concept of grid 'activation'. Once activated, all downstream tasks apply to the set of variables that exist on that grid.

NetCDF sources with numeric types are chosen by default, even if existing 'NC_CHAR' type variables are on the largest grid. When read any 'NC_CHAR' type variables are exploded into single character elements so that dimensions match the source.

Value

a `tidync` object

Multi-source

When `x` is a character vector of length > 1 and `concat_dim` is specified, tidync builds a consolidated view across all sources. The first source is used as a template for grid structure, and the `concat_dim` coordinate values are read from each source and concatenated. Downstream operations (`hyper_filter`, `hyper_array`, etc.) work transparently across the collection.

Use `fast = TRUE` for large collections to skip full metadata validation of sources 2..N (only the `concat_dim` coordinate is read). Mismatches are detected lazily at data-read time.

For maximum speed, supply coordinate values directly via the list form `concat_dim = list(name = "time", values = <ve`. This opens only the first source (for the template) and builds the concat transform from the supplied values with zero additional file I/O. This is ideal when coordinate values are already available from a file database.

Grids

A grid is an instance of a particular set of dimensions, which can be shared by more than one variable. This is not the 'rank' of a variable (the number of dimensions) since a single data set may have many 3D variables composed of different sets of axes/dimensions. There's no formality around the concept of 'shape', as far as we know.

A dimension may have length zero, but this is a special case for a "measure" dimension, we think. (It doesn't mean the product of the dimensions is zero, for example).

Limitations

Files with compound types are not yet supported and should fail gracefully. Groups are not yet supported.

We haven't yet explored 'HDF5' in detail, so any feedback is appreciated. Major use of compound types is made by https://github.com/sosoc/croc.

Examples

```
## a SeaWiFS (S) Level-3 Mapped (L3m) monthly (MO) chlorophyll-a (CHL)
## remote sensing product at 9km resolution (at the equator)
## from the NASA ocean colour group in NetCDF4 format (.nc)
## for 31 day period January 2008 (S20080012008031)
f <- "S20080012008031.L3m_MO_CHL_chlor_a_9km.nc"
l3file <- system.file("extdata/oceandata", f, package= "tidync")
## skip on Solaris
if (!tolower(Sys.info()[["sysname"]]) == "sunos") {
tnc <- tidync(l3file)
print(tnc)
```

```
}

## very simple Unidata example file, with one dimension
## Not run:
uf <- system.file("extdata/unidata", "test_hgroups.nc", package = "tidync")
recNum <- tidync(uf) |> hyper_tibble()
print(recNum)

## End(Not run)
## a raw grid of Southern Ocean sea ice concentration from IFREMER
## it is 12.5km resolution passive microwave concentration values
## on a polar stereographic grid, on 2 October 2017, displaying the
## "hole in the ice"
ifr <- system.file("extdata/ifremer", "20171002.nc", package = "tidync")
ifrnc <- tidync(ifr)
ifrnc |> hyper_tibble(select_var = "concentration")

## multi-source: concatenate files along a dimension
## Not run:
files <- c("sst_2020_01.nc", "sst_2020_02.nc", "sst_2020_03.nc")
tnc <- tidync(files, concat_dim = "time")
tnc
## filter and read across all sources transparently
tnc |> hyper_filter(time = time > 18300) |> hyper_tibble()

## fast mode for large collections (skips full metadata scan)
all_files <- list.files("daily/", pattern = "\\.nc$", full.names = TRUE)
tnc_fast <- tidync(all_files, concat_dim = "time", fast = TRUE)

## zero file I/O: supply values from a file database (e.g. raadfiles)
## only the first file is opened (for the template)
dates <- as.Date(c("2020-01-01", "2020-01-02", "2020-01-03"))
tnc_db <- tidync(files, concat_dim = list(name = "time", values = dates))
## filter directly on the values you supplied
tnc_db |> hyper_filter(time = time > as.Date("2020-01-01")) |> hyper_tibble()

## End(Not run)
```

Index

activate, 2
activate(), *3*, *15*
active *(*activate*)*, 2
active(), *3*
active<- *(*activate*)*, 2

dplyr, *7*
dplyr::filter(), *6*
dplyr::mutate(), *6*

hyper_array, 4, *14*
hyper_array(), *5*, *7–9*, *14*, *15*
hyper_dims *(*hyper_vars*)*, 11
hyper_filter, 6
hyper_filter(), *3–8*, *14*, *15*
hyper_grids *(*hyper_vars*)*, 11
hyper_grids(), *3*
hyper_slice *(*hyper_array*)*, 4
hyper_tbl_cube, 7
hyper_tbl_cube(), *3*, *5*, *9*
hyper_tibble, 8
hyper_tibble(), *3*, *5*, *8*, *9*
hyper_transforms, 10
hyper_transforms(), *5*
hyper_vars, 11, *13*, *15*

nc_get, 12

print, *11*, *15*
print(), *4*
print.tidync, 12
print.tidync_data, *5*, 14

tbl_cube, *7*
tidync, *5*, *6*, *13*, 15
tidync(), *4*, *14*
tidync_data, *4*
tidync_data *(*hyper_array*)*, 4
transforms, *7*