Package: rerddap (via r-universe)

July 3, 2026

Title General Purpose Client for 'ERDDAP™' Servers

Description General purpose R client for 'ERDDAP™' servers. Includes functions to search for 'datasets', get summary information on 'datasets', and fetch 'datasets', in either 'csv' or 'netCDF' format. 'ERDDAP™' information: <https://upwell.pfeg.noaa.gov/erddap/information.html>.

Version 1.3.0

Date 2026-06-26

License MIT + file LICENSE

URL https://docs.ropensci.org/rerddap/, https://github.com/ropensci/rerddap

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

LazyData true

Depends R (>= 4.00)

Encoding UTF-8

Language en-US

Imports crul (>= 0.7.4), dplyr (>= 0.5.0), data.table (>= 1.12.0), digest, hoardr (>= 0.5.2), jsonlite (>= 1.6), lubridate, methods, nanoparquet, ncdf4 (>= 1.16), tibble, utils, xml2 (>= 1.2.0)

Suggests knitr, plotly, rmarkdown

VignetteBuilder knitr

X-schema.org-applicationCategory Climate

X-schema.org-keywords earth, science, climate, precipitation, temperature, storm, buoy, NOAA

X-schema.org-isPartOf https://ropensci.org

Config/testthat/edition 3

Config/roxygen2/version 8.0.0.9000

NeedsCompilation no

Author Scott Chamberlain [aut], Ben Tupper [ctb], Salvador Jesús Fernández Bejarano [ctb], Roy Mendelssohn [cre, ctb]

Maintainer Roy Mendelssohn <roy.mendelssohn@noaa.gov>

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

Date/Publication 2026-07-03 03:00:02 UTC

RemoteUrl https://github.com/cran/rerddap

RemoteRef HEAD

RemoteSha 8a6a84efa5a664e5be2bd0ee918ff7d934d07cb2

Contents

`browse` *Browse a dataset webpage.*

Description

Browse a dataset webpage.

Usage

```
browse(x, url = eurl(), ...)
```

Arguments

`x`	datasetid or an object associated with a datasetid such `info()`, `griddap()` or `tabledap()`
`url`	A URL for an ERDDAP™ server. Default: https://upwell.pfeg.noaa.gov/erddap/ - See `eurl()` for more information
`...`	Further args passed on to `utils::browseURL` (must be a named parameter)

Value

if in interactive mode, opens a URL in your default browser; if not, then prints the URL in the console

Author(s)

Ben Tupper <btupper@bigelow.org>

Examples

```
## Not run:
if (interactive()) {
# browse by dataset_id
browse('erdATastnhday')

# browse info class
my_info <- info('erdATastnhday')
browse(my_info)

# browse tabledap class
my_tabledap <- tabledap('erdCalCOFIlrvsiz', fields=c('latitude','longitude','larvae_size',
   'itis_tsn'), 'time>=2011-10-25', 'time<=2011-10-31')
browse(my_tabledap)
}
## End(Not run)
```

cache_delete *Delete cached files*

Description

Delete cached files

Usage

```
cache_delete(x, force = FALSE)

cache_delete_all(force = FALSE)
```

Arguments

x	File names
force	(logical) Should files be force deleted? Default: FALSE

See Also

Other cache: cache_details(), cache_list(), cache_setup()

Examples

```
## Not run:
# delete files by name in cache
# cache_delete('9911750294a039b8b517c8bf288978ea.csv')
# cache_delete(c('9911750294a039b8b517c8bf288978ea.csv',
#                'b26825b6737da13d6a52c28c8dfe690f.csv'))

# You can delete from the output of griddap or tabledap fxns
## tabledap
(table_res <- tabledap('erdCinpKfmBT'))
cache_delete(table_res)

## griddap
(out <- info('erdQMekm14day'))
(grid_res <- griddap(out,
 time = c('2015-12-28','2016-01-01'),
 latitude = c(24, 23),
 longitude = c(88, 90)
))
cache_delete(grid_res)

## End(Not run)
```

cache_details *Get details of cached files*

Description

Get details of cached files

Usage

```
cache_details(x)
```

Arguments

x	File names

Details

Can be used to list details for all files, both .nc and .csv types, or details for just individual files of class `tabledap`, `griddap_nc`, and `griddap_csv`

See Also

Other cache: `cache_delete()`, `cache_list()`, `cache_setup()`

Examples

```
## Not run:
# List details for all cached files
cache_details()

## End(Not run)
```

cache_list *List cached files*

Description

List cached files

Usage

```
cache_list()
```

See Also

Other cache: `cache_delete()`, `cache_details()`, `cache_setup()`

Examples

```
## Not run:
# list files in cache
cache_list()

# List info for files
## download some data first
tabledap('erdCinpKfmBT')
griddap('erdVHNchlamday',
 time = c('2015-04-01','2015-04-10'),
 latitude = c(18, 21),
 longitude = c(-120, -119)
)

(x <- cache_list())
cache_details(x$nc[1])
cache_details(x$csv[1])
cache_details()

# delete files by name in cache
# cache_delete(x$nc[1])
# cache_delete(x$nc[2:3])

## End(Not run)
```

`cache_setup` *Setup cache path*

Description

Setup cache path

Usage

```
cache_setup(full_path = NULL, temp_dir = FALSE)

cache_info()
```

Arguments

`full_path`	(character) the full path to use for storing cached files.
`temp_dir`	(logical) if `TRUE` use a randomly assigned `tempdir` (and `full_path` is ignored), if `FALSE`, you can use `full_path`.

Details

On opening, by default a temporary directory is created for caching files. To have files cached elsewhere, give the full path of where to cache files. Adding `temp_dir = TRUE` will again use a temporary dirctory for cacheing.

Value

the full cache path, a directory (character)

See Also

Other cache: `cache_delete()`, `cache_details()`, `cache_list()`

Examples

```
## Not run:
# default path
cache_setup()

# you can define your own path
cache_setup(path = "foobar")

# set a tempdir - better for programming with to avoid prompt
cache_setup(temp_dir = TRUE)

# cache info
cache_info()

## End(Not run)
```

`colors` *cmocean colors The cmocean color palette by Kristen Thyng as implemented in the R package "oce"*

Description

str(colors) List of 13 $ viridis $ cdom $ chlorophyll $ density $ freesurface $ oxygen $ par $ phase $ salinity $ temperature $ turbidity $ velocity $ vorticity

Usage

```
colors
```

Format

An object of class `list` of length 13.

convert_time *Convert a UDUNITS compatible time to ISO time*

Description

Convert a UDUNITS compatible time to ISO time

Usage

```
convert_time(
  n = NULL,
  isoTime = NULL,
  units = "seconds since 1970-01-01T00:00:00Z",
  url = eurl(),
  method = "local",
  ...
)
```

Arguments

n	numeric; A unix time number.
isoTime	character; A string time representation.
units	character; Units to return. Default: "seconds since 1970-01-01T00:00:00Z"
url	Base URL of the ERDDAP™ server. See eurl() for more information
method	(character) One of local or web. Local simply uses as.POSIXct(), while web method uses the ERDDAP™ time conversion service /erddap/convert/time.txt
...	Curl options passed on to crul::verb-GET

Details

When method = "web" time zone is GMT/UTC

Examples

```
 ## Not run:
# local conversions
convert_time(n = 473472000)
convert_time(isoTime = "1985-01-02T00:00:00Z")

# using an ERDDAP™ web service
convert_time(n = 473472000, method = "web")
convert_time(isoTime = "1985-01-02T00:00:00Z", method = "web")

## End(Not run)
```

convert_units *Convert a CF Standard Name to/from a GCMD Science Keyword*

Description

Convert a CF Standard Name to/from a GCMD Science Keyword

Usage

```
convert_units(udunits = NULL, ucum = NULL, url = eurl(), ...)
```

Arguments

udunits	character; A UDUNITS character string https://www.unidata.ucar.edu/software/udunits/
ucum	character; A UCUM character string https://ucum.org/ucum.html
url	Base URL of the ERDDAP server. See `eurl()` for more information
...	Curl options passed on to crul::verb-GET

Examples

```
 ## Not run:
convert_units(udunits = "degree_C meter-1")
convert_units(ucum = "Cel.m-1")

## End(Not run)
```

disk *Options for saving ERDDAP™ datasets.*

Description

Options for saving ERDDAP™ datasets.

Usage

```
disk(path = NULL, overwrite = TRUE)

memory()
```

Arguments

path	Path to store files in. A directory, not a file. Default: the root cache path, see `cache_setup`
overwrite	(logical) Overwrite an existing file of the same name? Default: `TRUE`

ed_search *Search for ERDDAP™ tabledep or griddap datasets*

Description

Search for ERDDAP™ tabledep or griddap datasets

Usage

```
ed_search(
  query,
  page = NULL,
  page_size = NULL,
  which = "griddap",
  url = eurl(),
  ...
)

ed_datasets(which = "tabledap", url = eurl())
```

Arguments

query	(character) Search terms
page	(integer) Page number
page_size	(integer) Results per page
which	(character) One of tabledep or griddap.
url	A URL for an ERDDAP™ server. Default: https://upwell.pfeg.noaa.gov/erddap/ - See `eurl()` for more information
...	Curl options passed on to crul::verb-GET (must be named parameters)

References

https://upwell.pfeg.noaa.gov/erddap/index.html

Examples

```
## Not run:
(out <- ed_search(query='temperature'))
out$alldata[[1]]
(out <- ed_search(query='size'))
out$info

# List datasets
ed_datasets('table')
ed_datasets('grid')

# use a different ERDDAP™ server
```

```
## Marine Institute (Ireland)
ed_search("temperature", url = "http://erddap.marine.ie/erddap/")

## End(Not run)
```

---

ed_search_adv *Advanced search for ERDDAP™ tabledep or griddap datasets*

Description

Advanced search for ERDDAP™ tabledep or griddap datasets

Usage

```
ed_search_adv(
  query = NULL,
  page = 1,
  page_size = 1000,
  protocol = NULL,
  cdm_data_type = NULL,
  institution = NULL,
  ioos_category = NULL,
  keywords = NULL,
  long_name = NULL,
  standard_name = NULL,
  variableName = NULL,
  maxLat = NULL,
  minLon = NULL,
  maxLon = NULL,
  minLat = NULL,
  minTime = NULL,
  maxTime = NULL,
  url = eurl(),
  ...
)
```

Arguments

query	(character) Search terms
page	(integer) Page number. Default: 1
page_size	(integer) Results per page: Default: 1000
protocol	(character) One of any (default), tabledep or griddap
cdm_data_type	(character) One of grid, other, point, profile, timeseries, timeseriesprofile, trajectory, trajectoryprofile
institution	(character) An institution. See the dataset `institutions`
ioos_category	(character) An ioos category See the dataset `ioos_categories`

keywords	(character) A keywords. See the dataset keywords
long_name	(character) A long name. See the dataset longnames
standard_name	(character) A standar dname. See the dataset standardnames
variableName	(character) A variable name. See the dataset variablenames
minLon, maxLon	(numeric) Minimum and maximum longitude. Some datasets have longitude values within -180 to 180, others use 0 to 360. If you specify min and max Longitude within -180 to 180 (or 0 to 360), ERDDAP™ will only find datasets that match the values you specify. Consider doing one search: longitude -180 to 360, or two searches: longitude -180 to 180, and 0 to 360.
minLat, maxLat	(numeric) Minimum and maximum latitude, between -90 and 90
minTime, maxTime	(numeric/character) Minimum and maximum time. Time string with the format "yyyy-MM-ddTHH:mm:ssZ, (e.g., 2009-01-21T23:00:00Z). If you specify something, you must include at least yyyy-MM-dd; you can omit Z, :ss, :mm, :HH, and T. Always use UTC (GMT/Zulu) time. Or specify the number of seconds since 1970-01-01T00:00:00Z.
url	A URL for an ERDDAP™ server. Default: https://upwell.pfeg.noaa.gov/erddap/ - See eurl() for more information
...	Curl options passed on to crul::verb-GET (must be named parameters)

References

https://upwell.pfeg.noaa.gov/erddap/index.html

Examples

```
## Not run:
ed_search_adv(query = 'temperature')
ed_search_adv(query = 'temperature', protocol = "griddap")
ed_search_adv(query = 'temperature', protocol = "tabledap")
ed_search_adv(maxLat = 63, minLon = -107, maxLon = -87, minLat = 50,
  protocol = "griddap")
ed_search_adv(maxLat = 63, minLon = -107, maxLon = -87, minLat = 50,
  protocol = "tabledap")
ed_search_adv(minTime = "2010-01-01T00:00:00Z",
  maxTime="2010-02-01T00:00:00Z")
(out <- ed_search_adv(maxLat = 63, minLon = -107, maxLon = -87, minLat = 50,
             minTime = "2010-01-01T00:00:00Z",
             maxTime="2010-02-01T00:00:00Z"))
out$alldata[[1]]
ed_search_adv(variableName = 'upwelling')
ed_search_adv(query = 'upwelling', protocol = "tabledap")

# use a different URL
ed_search_adv(query = 'temperature', url = servers()$url[6])

## End(Not run)
```

`estimate_griddap_size` *Estimate the size of a rerddap::griddap() request*

Description

Uses coordinate metadata from an ERDDAP info() object to estimate how many grid cells will be returned and the total uncompressed byte count for each requested data variable. No network request is made.

Usage

```
estimate_griddap_size(
  info,
  ...,
  fields = "all",
  stride = 1L,
  spacing = list(),
  verbose = TRUE
)
```

Arguments

`info`	An object returned by rerddap::info().
`...`	Named dimension constraints, one per coordinate variable to constrain. Names must exactly match the coordinate variable names in the dataset (as returned by dimvars(info)). Each value is c(min, max); for time use ISO 8601 strings.
`fields`	Character vector of data variable names, or "all" (default).
`stride`	Integer scalar or named list of per-dimension stride values, using the same convention as griddap(). Default 1.
`spacing`	Optional named list to override auto-detected spacing for one or more dimensions. For time, supply seconds as time_sec (e.g. spacing = list(time_sec = 86400)). For other dimensions use the coordinate variable name (e.g. spacing = list(latitude = 0.01, xi_rho = 1)).
`verbose`	Logical; print a formatted summary (default TRUE).

Details

Coordinate dimension names are taken directly from the info object using the same logic as rerddap:::dimvars() — the set-difference between all keys in info$alldata and the data variable names plus "NC_GLOBAL". This means any coordinate system works: geographic lat/lon, projected x/y, sigma-layer depth, ROMS xi_rho/eta_rho, etc.

Dimension constraints are passed via ... using the exact coordinate variable names reported by the dataset, the same way griddap() accepts them. Each constraint is a numeric (or character for time) vector of length 2: c(min, max).

Spacing is resolved in this order for each dimension:

1. User-supplied override in the spacing argument.
2. For time: (t_max - t_min) / (nValues - 1) derived from the dimension's nValues row and actual_range attribute. This correctly handles running composites where time steps are daily even though the composite window is e.g. 8 days.
3. NC_GLOBAL attributes: geospatial_lat_resolution, geospatial_lon_resolution, time_coverage_resolution (ISO 8601).
4. Coordinate variable attributes: point_spacing, resolution, spacing.
5. For time: averageSpacing string from the nValues row as a last resort.
6. If the constraint min == max the dimension contributes 1 point.
7. Otherwise: NA — a warning is issued and 1 point is assumed.

Value

Invisibly, a named list containing per-dimension point counts, spacing values, per-variable byte estimates, and total bytes.

Examples

```
## Not run:
  library(rerddap)

  myURL <- "https://coastwatch.pfeg.noaa.gov/erddap/"
  response <- try(httr::HEAD(myURL, httr::timeout(10)), silent = TRUE)
  if (inherits(response, "try-error")) {
     stop("The ERDDAP\u2122 server is not responding")
  }
  info <- rerddap::info("erdMH1chla8day", url = myURL)
  estimate_griddap_size(info,
    latitude  = c(30, 50),
    longitude = c(-140, -110),
    time      = c("2020-01-01", "2020-12-31"))

## End(Not run)
```

---

eurl *Default ERDDAP™ server URL*

Description

Default ERDDAP™ server URL

Usage

```
eurl()
```

Details

default url is https://upwell.pfeg.noaa.gov/erddap/

You can set a default using an environment variable so you don't have to pass anything to the URL parameter in your function calls.

In your .Renviron file or similar set a URL for the environment variable `RERDDAP_DEFAULT_URL`, like `RERDDAP_DEFAULT_URL=https://upwell.pfeg.noaa.gov/erddap/`

It's important that you include a trailing slash in your URL

Examples

```
eurl()
Sys.setenv(RERDDAP_DEFAULT_URL = "https://google.com")
Sys.getenv("RERDDAP_DEFAULT_URL")
eurl()
Sys.unsetenv("RERDDAP_DEFAULT_URL")
eurl()
```

`fipscounty` *Convert a FIPS County Code to/from a County Name*

Description

Convert a FIPS County Code to/from a County Name

Usage

```
fipscounty(county = NULL, code = NULL, url = eurl(), ...)
```

Arguments

`county`	character; A county name.
`code`	numeric; A FIPS code.
`url`	A URL for an ERDDAP™ server. Default: https://upwell.pfeg.noaa.gov/erddap/ - See `eurl()` for more information
`...`	Curl options passed on to crul::verb-GET

Examples

```
 ## Not run:
fipscounty(code = "06053")
fipscounty(county = "CA, Monterey")
fipscounty(county = "OR, Multnomah")

## End(Not run)
```

global_search *global_search*

Description

Search for ERDDAP™ tabledap or griddap datasets from a list of ERDDAP™ servers based on search terms.

Usage

```
global_search(query, server_list, which_service)
```

Arguments

query	(character) Search terms
server_list	(list of character) List of ERDDAP™ servers to search
which_service	(character) One of tabledep or griddap.

Details

Uses the 'reddap' function ed_search() to search over the list of servers

Value

If successful a dataframe wih columns:

- title - the dataset title
- dataset_id - the datasetid on that ERDDAP™ server
- url - base url of dataset ERDDAP™ server

if urls are valid, no match is found, will return no match found else returns error message

See Also

HttpClient

Examples

```
# get list of servers know by
# https://irishmarineinstitute.github.io/awesome-erddap
# e_servers <- servers()$url
# select a couple to search
# e_servers <- e_servers[c(1, 40)]
# to meet CRAN time limits will only search 1 place
e_servers <- "https://coastwatch.pfeg.noaa.gov/erddap/"
test_query <- 'C-HARM v1 2-Day Forecast'
query_results <- global_search(test_query, e_servers, "griddap")
```

griddap *Get ERDDAP(TM) gridded data*

Description

Get ERDDAP(TM) gridded data

Usage

```
griddap(
  datasetx,
  ...,
  fields = "all",
  stride = 1,
  fmt = "nc",
  url = eurl(),
  store = disk(),
  read = TRUE,
  callopts = list()
)
```

Arguments

datasetx	Anything coercable to an object of class info. So the output of a call to info, or a datasetid, which will internally be passed through info
...	Dimension arguments. See examples. Can be any 1 or more of the dimensions for the particular dataset - and the dimensions vary by dataset. For each dimension, pass in a vector of length two, with min and max value desired. at least 1 required.
fields	(character) Fields to return, in a character vector.
stride	(integer) How many values to get. 1 = get every value, 2 = get every other value, etc. Default: 1 (i.e., get every value)
fmt	(character) One of nc (for netcdf), csv, or parquet. Default: nc
url	A URL for an ERDDAP™ server. Default: https://upwell.pfeg.noaa.gov/erddap/ - See eurl() for more information
store	One of disk (default) or memory. You can pass options to disk. Beware: if you choose fmt="nc", we force store=disk() because nc files have to be written to disk.
read	(logical) Read data into memory or not. Does not apply when store parameter is set to memory (which reads data into memory). For large csv, or especially netcdf files, you may want to set this to FALSE, which simply returns a summary of the dataset - and you can read in data piecemeal later. Default: TRUE
callopts	Curl options passed on to verb-GET

Details

Details:

If you run into an error like "HTTP Status 500 - There was a (temporary?) problem. Wait a minute, then try again.". it's likely they are hitting up against a size limit, and they should reduce the amount of data they are requesting either via space, time, or variables. Pass in `config = verbose()` to the request, and paste the URL into your browser to see if the output is garbled to examine if there's a problem with servers or this package

Value

An object of class `griddap_csv` if csv chosen or `griddap_nc` if nc file format chosen.

- `griddap_csv`: a data.frame created from the downloaded csv data
- `griddap_nc`: a list, with slots "summary" and "data". "summary" is the unclassed output from `ncdf4::nc_open`, from which you can do any netcdf operations you like. "data" is a data.frame created from the netcdf data. the data.frame may be empty if there were problems parsing the netcdf data

Both have the attributes: datasetid (the dataset id), path (the path on file for the csv or nc file), url (the url requested to the ERDDAP server)

If `read=FALSE`, the data.frame for `griddap_csv` and the data.frame in the "data" slot is empty for `griddap_nc`

Dimensions and Variables

ERDDAP™ grid dap data has this concept of dimenions vs. variables. Dimensions are things like time, latitude, longitude, altitude, and depth. Whereas variables are the measured variables, e.g., temperature, salinity, air.

You can't separately adjust values for dimensions for different variables. So, here's how it's gonna work:

Pass in lower and upper limits you want for each dimension as a vector (e.g., `c(1,2)`), or leave to defaults (i.e., don't pass anything to a dimension). Then pick which variables you want returned via the `fields` parameter. If you don't pass in options to the `fields` parameter, you get all variables back.

To get the dimensions and variables, along with other metadata for a dataset, run `info`, and each will be shown, with their min and max values, and some other metadata.

Where does the data go?

You can choose where data is stored. Be careful though. You can easily get a single file of hundreds of MB's (upper limit: 2 GB) in size with a single request. To the `store` parameter, pass `memory` if you want to store the data in memory (saved as a data.frame), or pass `disk` if you want to store on disk in a file. Note that `memory` and `disk` are not character strings, but function calls. `memory` does not accept any inputs, while `disk` does. Possibly will add other options, like "sql" for storing in a SQL database.

Non-lat/lon grid data

Some gridded datasets have latitude/longitude components, but some do not. When nc format gridded datasets have latitude and longitude we "melt" them into a data.frame for easy downstream consumption. When nc format gridded datasets do not have latitude and longitude components, we do not read in the data, throw a warning saying so. You can readin the nc file yourself with the file path. CSV format is not affected by this issue as CSV data is easily turned into a data.frame regardless of whether latitude/longitude data are present.

References

https://upwell.pfeg.noaa.gov/erddap/rest.html

Examples

```
## Not run:
# single variable dataset
## You can pass in the outpu of a call to info
(out <- info('erdVHNchlamday'))
## Or, pass in a dataset id
(res <- griddap('erdVHNchlamday',
 time = c('2015-04-01','2015-04-10'),
 latitude = c(18, 21),
 longitude = c(-120, -119)
))

# multi-variable dataset
(out <- info('erdQMekm14day'))
(res <- griddap(out,
 time = c('2015-12-28','2016-01-01'),
 latitude = c(24, 23),
 longitude = c(88, 90)
))
(res <- griddap(out, time = c('2015-12-28','2016-01-01'),
   latitude = c(24, 23), longitude = c(88, 90), fields = 'mod_current'))
(res <- griddap(out, time = c('2015-12-28','2016-01-01'),
   latitude = c(24, 23), longitude = c(88, 90), fields = 'mod_current',
   stride = c(1,2,1,2)))
(res <- griddap(out, time = c('2015-12-28','2016-01-01'),
   latitude = c(24, 23), longitude = c(88, 90),
   fields = c('mod_current','u_current')))


# Write to memory (within R), or to disk
(out <- info('erdQSwindmday'))
## disk, by default (to prevent bogging down system w/ large datasets)
## you can also pass in path and overwrite options to disk()
(res <- griddap(out,
 time = c('2006-07-11','2006-07-20'),
 longitude = c(166, 170),
 store = disk()
))
```

```
## the 2nd call is much faster as it's mostly just the time of reading in
## the table from disk
system.time( griddap(out,
 time = c('2006-07-11','2006-07-15'),
 longitude = c(10, 15),
 store = disk()
) )
system.time( griddap(out,
 time = c('2006-07-11','2006-07-15'),
 longitude = c(10, 15),
 store = disk()
) )

## memory - you have to choose fmt="csv" if you use memory
(res <- griddap("erdMBchla1day",
 time = c('2015-01-01','2015-01-03'),
 latitude = c(14, 15),
 longitude = c(125, 126),
 fmt = "csv", store = memory()
))

## Use ncdf4 package to parse data
info("erdMBchla1day")
(res <- griddap("erdMBchla1day",
 time = c('2015-01-01','2015-01-03'),
 latitude = c(14, 15),
 longitude = c(125, 126)
))

# Get data in csv format
## by default, we get netcdf format data
(res <- griddap('erdMBchla1day',
 time = c('2015-01-01','2015-01-03'),
 latitude = c(14, 15),
 longitude = c(125, 126),
 fmt = "csv"
))

# Use a different ERDDAP™ server url
## NOAA IOOS PacIOOS
url = "https://cwcgom.aoml.noaa.gov/erddap/"
out <- info("miamiacidification", url = url)
(res <- griddap(out,
 time = c('2019-11-01','2019-11-03'),
 latitude = c(15, 16),
 longitude = c(-90, -88)
))
## pass directly into griddap() - if you pass a datasetid string directly
## you must pass in the url or you'll be querying the default ERDDAP™ url,
## which isn't the one you want if you're not using the default ERDDAP™ url
griddap("miamiacidification", url = url,
 time = c('2019-11-01','2019-11-03'),
 latitude = c(15, 16),
```

```
 longitude = c(-90, -88)
)

# Using 'last'
## with time
griddap('erdVHNchlamday',
 time = c('last-5','last'),
 latitude = c(18, 21),
 longitude = c(-120, -119)
)
## with latitude
griddap('erdVHNchlamday',
  time = c('2015-04-01','2015-04-10'),
  latitude = c('last', 'last'),
  longitude = c(-120, -119)
)
## with longitude
griddap('erdVHNchlamday',
  time = c('2015-04-01','2015-04-10'),
  latitude = c(18, 21),
  longitude = c('last', 'last')
)

# datasets without lat/lon grid and with fmt=nc
# FIXME: this dataset is gone
# (x <- info('glos_tds_5912_ca66_3f41'))
# res <- griddap(x,
#   time = c('2018-04-01','2018-04-10'),
#   ny = c(1, 2),
#   nx = c(3, 5)
# )
## data.frame is empty
# res$data
## read in from the nc file path
# ncdf4::nc_open(res$summary$filename)

## End(Not run)
```

`info` *Get information on an ERDDAP(TM) dataset.*

Description

Get information on an ERDDAP(TM) dataset.

Usage

```
info(datasetid, url = eurl(), ...)

as.info(x, url)
```

Arguments

`datasetid`	Dataset id
`url`	A URL for an ERDDAP(TM) server. Default: https://upwell.pfeg.noaa.gov/erddap/ - See `eurl()` for more information
`...`	Further args passed on to crul::verb-GET (must be a named parameter)
`x`	A datasetid or the output of `info`

Value

Prints a summary of the data on return, but you can index to various information.

The data is a list of length two with:

- variables - Data.frame of variables and their types
- alldata - List of data variables and their full attributes

Where `alldata` element has many data.frame's, one for each variable, with metadata for that variable. E.g., for griddap dataset `noaa_pfeg_696e_ec99_6fa6`, `alldata` has:

- NC_GLOBAL
- time
- latitude
- longitude
- sss

References

https://upwell.pfeg.noaa.gov/erddap/index.html

Examples

```
## Not run:
# grid dap datasets
info('erdATastnhday')

(out <- ed_search(query='temperature'))
info(out$info$dataset_id[5])
info(out$info$dataset_id[15])
info(out$info$dataset_id[25])
info(out$info$dataset_id[150])
info(out$info$dataset_id[400])
info(out$info$dataset_id[678])

out <- info(datasetid='erdMBchla1day')
## See brief overview of the variables and range of possible values, if given
out$variables
## all information on longitude
out$alldata$longitude
## all information on chlorophyll
```

```
out$alldata$chlorophyll

# table dap datasets
(out <- ed_search(query='temperature', which = "table"))
info(out$info$dataset_id[1])
info(out$info$dataset_id[2])
info(out$info$dataset_id[3])
info(out$info$dataset_id[4])

info('erdCinpKfmBT')
out <- info('erdCinpKfmBT')
## See brief overview of the variables and range of possible values, if given
out$variables
## all information on longitude
out$alldata$longitude
## all information on Haliotis_corrugata_Mean_Density
out$alldata$Haliotis_corrugata_Mean_Density

# use a different ERDDAP(TM) server
## Marine Institute (Ireland)
info("IMI_CONN_2D", url = "http://erddap.marine.ie/erddap/")

## End(Not run)
```

`institutions` *institutions*

Description

institutions

Format

A character vector

`ioos_categories` *ioos_categories*

Description

ioos_categories

Format

A character vector

key_words *Convert a CF Standard Name to/from a GCMD Science Keyword*

Description

Convert a CF Standard Name to/from a GCMD Science Keyword

Usage

```
key_words(cf = NULL, gcmd = NULL, url = eurl(), ...)
```

Arguments

cf	character; A cf standard name http://cfconventions.org/Data/cf-standard-names/27/build/cf-standard-name-table.html
gcmd	character; A GCMD science keyword http://gcmd.gsfc.nasa.gov/learn/keyword_list.html
url	A URL for an ERDDAP™ server. Default: https://upwell.pfeg.noaa.gov/erddap/. See `eurl()` for more information
...	Curl options passed on to crul::verb-GET

Examples

```
 ## Not run:
key_words(cf = "air_pressure")
cat(key_words(cf = "air_pressure"))

# a different ERDDAP™ server
# key_words(cf = "air_pressure", url = servers()$url[6])

## End(Not run)
```

keywords *keywords*

Description

keywords

Format

A character vector

longnames *longnames*

Description

longnames

Format

A character vector

servers *ERDDAP™ server URLS and other info*

Description

ERDDAP™ server URLS and other info

Usage

```
servers(...)
```

Arguments

...	curl options passed on to crul::verb-GET

Value

data.frame with 3 columns:

- name (character): ERDDAP™ name
- url (character): ERDDAP™ url
- public (logical): whether it's public or not

Examples

```
## Not run:
servers()

## End(Not run)
```

standardnames *standardnames*

Description

standardnames

Format

A character vector

tabledap *Get ERDDAP™ tabledap data.*

Description

Get ERDDAP™ tabledap data.

Usage

```
tabledap(
  x,
  ...,
  fields = NULL,
  distinct = FALSE,
  orderby = NULL,
  orderbymax = NULL,
  orderbymin = NULL,
  orderbyminmax = NULL,
  units = NULL,
  fmt = "csv",
  url = eurl(),
  store = disk(),
  callopts = list()
)
```

Arguments

x	Anything coercable to an object of class info. So the output of a call to info(), or a datasetid, which will internally be passed through info()
...	Any number of key-value pairs in quotes as query constraints. See Details & examples
fields	Columns to return, as a character vector

distinct	If TRUE ERDDAP™ will sort all of the rows in the results table (starting with the first requested variable, then using the second requested variable if the first variable has a tie, ...), then remove all non-unique rows of data. In many situations, ERDDAP™ can return distinct values quickly and efficiently. But in some cases, ERDDAP™ must look through all rows of the source dataset.
orderby	If used, ERDDAP™ will sort all of the rows in the results table (starting with the first variable, then using the second variable if the first variable has a tie, ...). Normally, the rows of data in the response table are in the order they arrived from the data source. orderBy allows you to request that the results table be sorted in a specific way. For example, use orderby=c("stationID,time") to get the results sorted by stationID, then time. The orderby variables MUST be included in the list of requested variables in the fields parameter.
orderbymax	Give a vector of one or more fields, that must be included in the fields parameter as well. Gives back data given constraints. ERDDAP™ will sort all of the rows in the results table (starting with the first variable, then using the second variable if the first variable has a tie, ...) and then just keeps the rows where the value of the last sort variable is highest (for each combination of other values).
orderbymin	Same as orderbymax parameter, except returns minimum value.
orderbyminmax	Same as orderbymax parameter, except returns two rows for every combination of the n-1 variables: one row with the minimum value, and one row with the maximum value.
units	One of 'udunits' (units will be described via the UDUNITS standard (e.g.,degrees_C)) or 'ucum' (units will be described via the UCUM standard (e.g., Cel)).
fmt	whether download should be as '.csv' (default) or '.parquet'
url	A URL for an ERDDAP™ server. Default: https://upwell.pfeg.noaa.gov/erddap/ - See eurl() for more information
store	One of disk (default) or memory. You can pass options to disk
callopts	Curl options passed on to crul::verb-GET (must be named parameters)

Details

For key-value pair query constraints, the valid operators are =, != (not equals), =~ (a regular expression test), <, <=, >, and >= . For regular expressions you need to add a regular expression. For others, nothing more is needed. Construct the entry like 'time>=2001-07-07' with the parameter on the left, value on the right, and the operator in the middle, all within a set of quotes. Since ERDDAP accepts values other than =, we can't simply do time = '2001-07-07' as we normally would.

Server-side functionality: Some tasks are done server side. You don't have to worry about what that means. They are provided via parameters in this function. See distinct, orderby, orderbymax, orderbymin, orderbyminmax, and units.

Data is cached based on all parameters you use to get a dataset, including base url, query parameters. If you make the same exact call in the same or a different R session, as long you don't clear the cache, the function only reads data from disk, and does not have to request the data from the web again.

If you run into an error like "HTTP Status 500 - There was a (temporary?) problem. Wait a minute, then try again.". it's likely they are hitting up against a size limit, and they should reduce the amount

of data they are requesting either via space, time, or variables. Pass in `config = verbose()` to the request, and paste the URL into your browser to see if the output is garbled to examine if there's a problem with servers or this package

Value

An object of class `tabledap`. This class is a thin wrapper around a data.frame, so the data you get back is a data.frame with metadata attached as attributes (datasetid, path (path where the csv is stored on your machine), url (url for the request))

References

https://upwell.pfeg.noaa.gov/erddap/index.html

Examples

```
## Not run:
# Just passing the datasetid without fields gives all columns back
tabledap('erdCinpKfmBT')

# Pass time constraints
tabledap('erdCinpKfmBT', 'time>=2006-08-24')

# Pass in fields (i.e., columns to retrieve) & time constraints
tabledap('erdCinpKfmBT',
  fields = c('longitude', 'latitude', 'Aplysia_californica_Mean_Density'),
  'time>=2006-08-24'
)

# Get info on a datasetid, then get data given information learned
info('erdCalCOFIlrvsiz')$variables
tabledap('erdCalCOFIlrvsiz', fields=c('latitude','longitude','larvae_size',
   'itis_tsn'), 'time>=2011-10-25', 'time<=2011-10-31')

# An example workflow
## Search for data
(out <- ed_search(query='fish', which = 'table'))
## Using a datasetid, search for information on a datasetid
id <- out$alldata[[1]]$dataset_id
vars <- info(id)$variables
## Get data from the dataset
vars$variable_name[1:3]
tabledap(id, fields = vars$variable_name[1:3])

# Time constraint
## Limit by time with date only
(info <- info('erdCinpKfmBT'))
tabledap(info, fields = c(
  'latitude','longitude','Haliotis_fulgens_Mean_Density'),
  'time>=2001-07-14')

# Use distinct parameter - compare to distinct = FALSE
```

```
tabledap('sg114_3',
   fields=c('longitude','latitude','trajectory'),
   'time>=2008-12-05', distinct = TRUE)

# Use units parameter
## In this example, values are the same, but sometimes they can be different
## given the units value passed
tabledap('erdCinpKfmT', fields=c('longitude','latitude','time','temperature'),
   'time>=2007-09-19', 'time<=2007-09-21', units='udunits')
tabledap('erdCinpKfmT', fields=c('longitude','latitude','time','temperature'),
   'time>=2007-09-19', 'time<=2007-09-21', units='ucum')

# Use orderby parameter
tabledap('erdCinpKfmT', fields=c('longitude','latitude','time','temperature'),
   'time>=2007-09-19', 'time<=2007-09-21', orderby='temperature')
# Use orderbymax parameter
tabledap('erdCinpKfmT', fields=c('longitude','latitude','time','temperature'),
   'time>=2007-09-19', 'time<=2007-09-21', orderbymax='temperature')
# Use orderbymin parameter
tabledap('erdCinpKfmT', fields=c('longitude','latitude','time','temperature'),
   'time>=2007-09-19', 'time<=2007-09-21', orderbymin='temperature')
# Use orderbyminmax parameter
tabledap('erdCinpKfmT', fields=c('longitude','latitude','time','temperature'),
   'time>=2007-09-19', 'time<=2007-09-21', orderbyminmax='temperature')
# Use orderbymin parameter with multiple values
tabledap('erdCinpKfmT',
   fields=c('longitude','latitude','time','depth','temperature'),
   'time>=2007-06-10', 'time<=2007-09-21',
   orderbymax=c('depth','temperature')
)

# Integrate with taxize
out <- tabledap('erdCalCOFIlrvcntHBtoHI',
   fields = c('latitude','longitude','scientific_name','itis_tsn'),
   'time>=2007-06-10', 'time<=2007-09-21'
)
tsns <- unique(out$itis_tsn[1:100])
library("taxize")
classif <- classification(tsns, db = "itis")
head(rbind(classif)); tail(rbind(classif))

# Write to memory (within R), or to disk
(out <- info('erdCinpKfmBT'))
## disk, by default (to prevent bogging down system w/ large datasets)
## the 2nd call is much faster as it's mostly just the time of reading
## in the table from disk
system.time( tabledap('erdCinpKfmBT', store = disk()) )
system.time( tabledap('erdCinpKfmBT', store = disk()) )
## memory
tabledap('erdCinpKfmBT', store = memory())

# use a different ERDDAP™ server
## NOAA IOOS NERACOOS
```

```
url <- "http://www.neracoos.org/erddap/"
tabledap("E01_optics_hist", url = url)

## End(Not run)
```

`variablenames` *variablenames*

Description

variablenames

Format

A character vector

`version` *Get ERDDAP™ version*

Description

Get ERDDAP™ version

Usage

```
version(url = eurl(), ...)
```

Arguments

`url`	A URL for an ERDDAP™ server. Default: https://upwell.pfeg.noaa.gov/erddap/ - See `eurl()` for more information
`...`	Curl options passed on to crul::verb-GET

Examples

```
## Not run:
version()
ss <- servers()
version(ss$url[2])
version(ss$url[3])

## End(Not run)
```

Index