# Package ‘rerddap’

October 14, 2022

**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](https://upwell.pfeg.noaa.gov/erddap/information.html).

**Version**  1.0.0

**Date**  2022-10-03

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

**Suggests**  knitr, rmarkdown

**RoxygenNote**  7.2.1

**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

**NeedsCompilation**  no

**Author**  Scott Chamberlain [aut], Ben Tupper [ctb], Roy Mendelssohn [cre, ctb]
rerddap-package

Maintainer  Roy Mendelssohn <roy.mendelssohn@noaa.gov>
Repository  CRAN
Date/Publication  2022-10-03 15:30:02 UTC

R topics documented:

rerddap-package ........................................... 2
browse ........................................................ 3
cache_delete ............................................... 4
cache_details ............................................. 5
cache_list .................................................. 6
cache_setup ............................................... 7
colors ....................................................... 8
convert_time ............................................. 8
convert_units ............................................ 9
disk ........................................................ 10
ed_search ................................................... 10
ed_search_adv ............................................ 12
eurl .......................................................... 14
fipscounty .................................................. 14
global_search .............................................. 15
griddap ..................................................... 16
info ........................................................ 20
institutions ............................................... 22
ioos_categories ........................................... 23
keywords .................................................... 23
key_words ................................................... 23
longnames ................................................... 24
servers ...................................................... 24
standardnames ............................................ 25
tabledap ..................................................... 25
variablenames ............................................ 29
version ..................................................... 29

Index  30

rerddap-package  rerddap

Description

General purpose R client for ERDDAP servers
ERDDAP info

NOAA’s ERDDAP service holds many datasets of interest. It’s built on top of OPenDAP. You can search for datasets via `ed_search()`, list datasets via `ed_datasets()`, get information on a single dataset via `info()`, then get data you want for either `tabledap()` type via `tabledap()`, or for `griddap()` type via `griddap()`

tabledap/griddap

tabledap and griddap have different interfaces to query for data, so `tabledap()` and `griddap()` are separated out as separate functions even though some of the internals are the same. In particular, with `tabledap` you can query on/subset all variables, whereas with `griddap`, you can only query on/subset the dimension variables (e.g., latitude, longitude, altitude).

Data size

With `griddap` data via `griddap()` you can get a lot of data quickly. Try small searches of a dataset to start to get a sense for the data, then you can increase the amount of data you get. See `griddap()` for more details.

Caching

`rerddap` by default caches the requests you make, so that if you happen to make the same request again, the data is restored from the cache, rather than having to go out and retrieve it remotely. For most applications, this is good, as it can speed things up when doing a lot of request in a script, and works because in most cases an ERDDAP request is "idempotent". This means that the the request will always return the same thing no matter what requests came before - it doesn’t depend on state. However this is not true if the script uses either "last" in `griddap()` or "now" in `tabledap()` as these will return different values as time elapses and data are added to the datasets. While it is desirable to have ERDDAP purely idempotent, the "last" and "now" constructs are very helpful for people using ERDDAP in dashboards, webpages, regular input to models and the like, and the benefits far outweigh the problems. However, if you are using either "last" or "now" in an `rerddap` based script, you want to be very careful to clear the `rerddap` cache, otherwise the request will be viewed as the same, and the data from the last request, rather than the latest data, will be returned.

---

**browse**

Browse a dataset webpage.

**Description**

Note that it is an error to call this when `base::interactive()` returns `FALSE`

**Usage**

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

x  datasetid or an object associated with a datasetid such as `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

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

```r

<table>
<thead>
<tr>
<th>cache_delete</th>
<th>Delete cached files</th>
</tr>
</thead>
</table>

Description

Delete cached files

Usage

```r
cache_delete(x, force = FALSE)
cache_delete_all(force = FALSE)
```
cache_details

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)
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_setup

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

```r
cache_setup(path_suffix = NULL, temp_dir = FALSE)
```

cache_info()

Arguments

- `path_suffix` (character) the path suffix to use for storing cached files, appended to user cache dir.
- `temp_dir` (logical) if TRUE use a randomly assigned tempdir (and path_suffix is ignored), if FALSE, you can use path_suffix.

Details
Looks first if the user has set a cache path suffix in an env var or R option. If not found, proceeds to use a temp directory if not in interactive mode, but if interactive, asks user to setup a default cache location that will work across sessions (but user can say no, in which case a temp directory will be used, and each package start will require cache setup again)

Value
the full cache path, a directory (character)

See Also
Other cache: `cache_delete()`, `cache_details()`, `cache_list()`
Examples

```r
## 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**

```r
str(colors)
```

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

---

**Usage**

```r
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
convert_units

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 `crl::verb-GET`

Details

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

Examples

```r
## 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

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

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 `crl::verb-GET`

Examples

```r
## 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
ed_search

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

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 standard name. See the dataset standardnames
variableName  (character) A variable name. See the dataset variablenames
minLon, maxLon (numeric) Minimum and maximum longitude. Some datasets have longitude
counting 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 for-
matter "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 sec-
onds since 1970-01-01T00:00:00Z.
url A URL for an ERDDAP server. Default: https://upwell.pfeg.noaa.gov/erddap/- See
... 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 = "Var temperature")
ed_search_adv(query = "Var temperature", protocol = "griddap")
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)
eur1  
*Default ERDDAP server URL*

**Description**

Default ERDDAP server URL

**Usage**

`eur1()`

**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**

`eur1()`

```r
Sys.setenv(RERDDAP_DEFAULT_URL = "https://google.com")
Sys.getenv("RERDDAP_DEFAULT_URL")
eur1()
Sys.unsetenv("RERDDAP_DEFAULT_URL")
eur1()
```

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 `eur1()` for more information
- `...` Curl options passed on to `crl::verb-GET`
global_search

Examples

## Not run:

```r
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 with 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

```r
# 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 <- 'NOAA/NCDC Blended Monthly'
query_results <- global_search(test_query, e_servers, "griddap")

---

griddap  Get ERDDAP gridded data

Description

Get ERDDAP gridded data

Usage

```r
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 csv or nc (for netcdf). 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 dimensions 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

```r
## Not run:
# single variable dataset
# You can pass in the output 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'),
  longitude = c(166, 170),
  store = memory()
))
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 dataset.
Description
Get information on an ERDDAP dataset.

Usage
info(datasetid, url = eurl(), ...)
as.info(x, url)

Arguments
datasetid          Dataset id
url                A URL for an ERDDAP server. Default: https://upwell.pfeg.noaa.gov/erddap/ -
                  See eurl() for more information
                  ... Further args passed on to curl::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 vari-
able. 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])
```r
info(out$info$dataset_id[150])
info(out$info$dataset_id[400])
info(out$info$dataset_id[678])

out <- info(datasetid='erdMBChlalday')
## 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 server
## Marine Institute (Ireland)
info("MI_CONN_2D", url = "http://erddap.marine.ie/erddap/")

## End(Not run)
```

---

**Description**

institutions

**Format**

A character vector
ioos_categories

Description
ioos_categories

Format
A character vector

keywords

Description
keywords

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

```r
## 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)
```

### longnames

**Description**

longnames

**Format**

A character vector

### servers

**Description**

ERDDAP server URLs and other info

**Usage**

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

**Arguments**

... curl options passed on to `crl::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

```r
## Not run:
servers()

## End(Not run)
```

---

**standardnames**

### Description

standardnames

### Format

A character vector

---

**tabledap**

*Get ERDDAP tabledap data.*

### Description

Get ERDDAP tabledap data.

### Usage

```r
tabledap(
x,
  ...,
  fields = NULL,
  distinct = FALSE,
  orderby = NULL,
  orderbymax = NULL,
  orderbymin = NULL,
  orderbyminmax = NULL,
  units = NULL,
  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)).
- **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**

```r
## 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
fields = c('longitude', 'latitude', 'Aplysia_californica_Mean_Density'),
          'time>=2006-08-24')

# Get info on a datasetid, then get data given information learned
info('erdCalCOF1rvsiz')$variables
tabledap('erdCalCOF1rvsiz', 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', 'Halioitis_fulgens_Mean_Density',
  'time' >= '2007-07-14')

# Use orderbymin parameter with multiple values
(out <- info('erdCinpKfmBT'))
tabledap(out, fields = c('latitude', 'longitude', 'Halioitis_fulgens_Mean_Density',
  'time' >= '2007-07-14')

# Write to memory (within R), or to disk
(head(rbind(classif)); tail(rbind(classif)))

library("taxize")

## the 2nd call is much faster as it's mostly just the time of reading
# in the table from disk

## Use orderbyminmax parameter

## Use orderbymin parameter

## In this example, values are the same, but sometimes they can be different
## given the units value passed

## Use units parameter

## Use distinct parameter - compare to distinct = FALSE

## Limit by time with date only

## Use orderby parameter

## Use orderbymax parameter

## Use orderbymin parameter

## Use orderbyminmax parameter

## Use orderbymin parameter with multiple values

# Integrate with taxize
out <- tabledap('erdCinpKfmBT',
  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))
```
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)
```

---

### 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 `crl::verb-GET`

#### Examples

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

## End(Not run)
```
Index

* cache
  cache_delete, 4
cache_details, 5
cache_list, 6
cache_setup, 7
* datasets
  colors, 8
  institutions, 22
  ioos_categories, 23
  keywords, 23
  longnames, 24
  standardnames, 25
  variablenames, 29
* package
  rerddap-package, 2
as.info (info), 20
as.POSIXct(), 9

browse, 3
cache_delete, 4, 6, 7
cache_delete_all (cache_delete), 4
cache_details, 5, 6, 7
cache_info (cache_setup), 7
cache_list, 5, 6, 7
cache_setup, 5, 6, 7, 10
colors, 8
convert_time, 8
convert_units, 9
crul::verb-GET, 9–11, 13, 14, 16, 21, 23, 24, 26, 29
disk, 10, 17, 18
ed_datasets (ed_search), 10
ed_datasets(), 3
ed_search, 10
ed_search(), 3
ed_search_adv, 12
eurl(), 4, 9–11, 13, 14, 16, 21, 23, 26, 29
fipscounty, 14
global_search, 15
griddap, 16
griddap(), 3, 4
HttpClient, 15
info, 16, 17, 20
info(), 3, 4, 26
institutions, 22
ioos_categories, 23
key_words, 23
keywords, 23
longnames, 24
memory, 17, 18
memory (disk), 10
rerddap (rerddap-package), 2
rerddap-package, 2
servers, 24
standardnames, 25
tabledap, 25
tabledap(), 3, 4
variablenames, 29
version, 29