Package ‘rnoaa’

February 20, 2015

Title  NOAA Climate Data from R

Description  An R client for many NOAA data sources including the NCDC climate API at http://www.ncdc.noaa.gov/cdo-web/webservices/v2, with functions for each of the API endpoints: data, data categories, data sets, data types, locations, location categories, and stations. In addition, we have an interface for NOAA sea ice data, the NOAA severe weather inventory, ERDDAP data, NOAA Historical Observing Metadata Repository (HOMR) data, NOAA storm data via IBTrACS, and tornado data via the NOAA storm prediction center. NOAA buoy data is only on the buoy branch in the Github repository for this package (see url below).

Version  0.3.3

Date  2014-12-19

License  MIT + file LICENSE

URL  https://github.com/ropensci/rnoaa

BugReports  http://www.github.com/ropensci/rnoaa/issues

LazyData  true

VignetteBuilder  knitr

Imports  httr, lubridate, plyr, ggplot2, scales, sp, RCurl, rgdal, rgeos, maptools, stringr, XML, data.table, jsonlite, digest

Suggests  testthat, roxygen2, knitr, taxize, dplyr

Author  Hart Edmund [aut],
Scott Chamberlain [aut, cre],
Karthik Ram [aut]

Maintainer  Scott Chamberlain <myrmecocystus@gmail.com>

NeedsCompilation  no

Repository  CRAN

Date/Publication  2014-12-20 15:46:42
Description

rnoaa is an R interface to NOAA climate data.

Details

Many functions in this package interact with the National Climatic Data Center application programming interface (API) at http://www.ncdc.noaa.gov/cdo-web/webservices/v2, all of which functions start with `ncdc_`. An access token, or API key, is required to use all the `ncdc_` functions. The key is required by NOAA, not the creators of this R package. Go to the link given above to get an API key.
More NOAA data sources are being added through time. Data sources and their function prefixes are:

- `buoy_*` - NOAA Buoy data, only on buoy branch in Github repo
- `erddap_*` - NOAA ERDDAP vignette
- `ncdc_*` - NOAA National Climatic Data Center (NCDC) vignette (examples)
- `seaice` - Sea ice vignette
- `swdi` - Severe Weather Data Inventory (SWDI) vignette
- `homr_*` - Historical Observing Metadata Repository (HOMR) vignette
- `storm_*` - Storms (IBTrACS) vignette
- `tornadoes` - From the NOAA Storm Prediction Center

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

---

**disk**

*Options for saving ERDDAP datasets.*

### Description

Options for saving ERDDAP datasets.

### Usage

```r
 disk(path = "~/.rnoaa/erddap", overwrite = FALSE)
```

### Arguments

- **path**
  - Path to store files in. Default: `~/.rnoaa/upwell`
- **overwrite**
  - (logical) Overwrite an existing file of the same name? Default: TRUE
erddap

ERDDAP Information

Description

NOAA’s ERDDAP service holds many datasets of interest. It’s built on top of OPenDAP [http://www.opendap.org/](http://www.opendap.org/). You can search for datasets via `erddap_search`, list datasets via `erddap_datasets`, get information on a single dataset via `erddap_info`, then get data you want for either tabledap type via `erddap_table`, or for griddap type via `erddap_grid`.

Details

tabledap and griddap have different interfaces to query for data, so `erddap_table` and `erddap_grid` 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).

**NOTE:** With griddap data via `erddap_grid` 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 `erddap_grid` for more details.

The following are the ERDDAP functions:

- `erddap_search`
- `erddap_datasets`
- `erddap_info`
- `erddap_table`
- `erddap_grid`

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

References

erddap_clear_cache

**Clear cached files**

**Description**

Clear cached files

**Usage**

erddap_clear_cache(path = disk())

**Arguments**

- `path` Path to location of cached files. Defaults to `disk()$path`

**Details**

BEWARE: this will clear all cached files.

erddap_grid

**Get ERDDAP griddap data.**

**Description**

Get ERDDAP griddap data.

**Usage**

erddap_grid(x, ..., fields = "all", stride = 1, store = disk(), callopts = list())

**Arguments**

- `x` Anything coercable to an object of class `erddap_info`. So the output of a call to `erddap_info`, or a datasetid, which will internally be passed through `erddap_info`.
- `...` Dimension arguments.
- `fields` Fields to return, 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)
- `store` One of `disk` (default) or memory. You can pass options to `disk`
- `callopts` Pass on curl options to `GET`

**Details**

Details:
Dimensions and Variables

ERDDAP grid dap data has this concept of dimensions vs. variables. So, dimensions are things like time, latitude, longitude, and altitude. Whereas variables are the measured variables, e.g., temperature, salinity, and 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 `erddap_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. With griddap data, you can easily get a single file of hundreds of MB’s or GB’s in size with a single request. Using `disk()` caches files based on the URL of the request you perform, which is the combination of all parameters passed in, so if you refine a query by certain fields, etc., you will cach a different file. If you choose `overwrite=TRUE` within the `disk()` function then you’ll force writing a new file to disk. When you use `memory()`, no files are cached, data is stored in R’s memory.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

References

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

See Also

`erddap_table erddap_clear_cache`

Examples

```r
## Not run:
# single variable dataset
## You can pass in the output of a call to erddap_info
(out <- erddap_info('noaa_esrl_027d_0fb5_5d38'))
(res <- erddap_grid(out, 
  time = c('2012-01-01','2012-06-12'),
  latitude = c(21, 18),
  longitude = c(-80, -75))
## Or, pass in a dataset id
(res <- erddap_grid(x='noaa_esrl_027d_0fb5_5d38',
  time = c('2012-01-01','2012-06-12'),
  latitude = c(21, 18),
  longitude = c(-80, -75))
```
longitude = c(-80, -75)
)

# multi-variable dataset
(out <- erddap_info('noaa_gfdl_5081_7d4a_7570'))
(res <- erddap_grid(out,
    time = c('2005-11-01','2006-01-01'),
    latitude = c(20, 21),
    longitude = c(10, 11)
))
(res <- erddap_grid(out, time = c('2005-11-01','2006-01-01'), latitude = c(20, 21),
    longitude = c(10, 11), fields = 'uo'))
(res <- erddap_grid(out, time = c('2005-11-01','2006-01-01'), latitude = c(20, 21),
    longitude = c(10, 11), fields = c('uo', 'so')))
(res <- erddap_grid(out, time = c('2005-09-01','2006-01-01'), latitude = c(20, 21),
    longitude = c(10, 11), fields = 'none'))

# multi-variable dataset
## this one also has a 0-360 longitude system, BLARGH!!
(out <- erddap_info('noaa_gfdl_3c96_7879_a9d3'))
(res <- erddap_grid(out,
    time = c('2005-11-01','2006-01-01'),
    latitude = c(20, 22),
    longitude = c(-80, -75)
))
(res <- erddap_grid(out, time = c('2005-11-01','2006-01-01'),
    latitude = c(20, 22),
    longitude = c(-80, -75),
    depth = c(5, 50)
))

# Write to memory (within R), or to disk
(out <- erddap_info('noaa_pfeg_e9ae_3356_22f8'))
## disk, by default (to prevent bogging down system w/ large datasets)
## you can also pass in path and overwrite options to disk()
(res <- erddap_grid(out,
    time = c('2012-06-01','2012-06-12'),
    latitude = c(20, 21),
    longitude = c(-80, -75),
    store = disk()
))
## the 2nd call is much faster as it's mostly just the time of reading in the table from disk
system.time( erddap_grid(out, time = c('2012-06-01','2012-06-12'),
    latitude = c(20, 21),
    longitude = c(-80, -75),
    store = disk()
)

system.time( erddap_grid(out, time = c('2012-06-01','2012-06-12'),
    latitude = c(20, 21),
    longitude = c(-80, -75),
    store = disk()
)

system.time( erddap_grid(out, time = c('2012-06-01','2012-06-12'),
    latitude = c(20, 21),
    longitude = c(-80, -75),
    store = disk()
)
erddap_info

Get information on an ERDDAP dataset.

Description
Get information on an ERDDAP dataset.

Usage
erddap_info(datasetid, ...) as.erddap_info(x)

Arguments
datasetid       Dataset id
...             Further args passed on to GET (must be a named parameter)
x               A datasetid or the output of erddap_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
Author(s)
Scott Chamberlain <myrmecocystus@gmail.com>

References
http://upwell.pfeg.noaa.gov/erddap/index.html

Examples

## Not run:
# grid dap datasets
erddap_info('noaa_pfeg_696e_ec99_6fa6')
erddap_info('noaa_ngdc_34bf_a95c_7e28')

(out <- erddap_search(query='temperature'))
erddap_info(out$info$dataset_id[5])
erddap_info(out$info$dataset_id[15])
erddap_info(out$info$dataset_id[25])
erddap_info(out$info$dataset_id[33])
erddap_info(out$info$dataset_id[65])
erddap_info(out$info$dataset_id[150])
erddap_info(out$info$dataset_id[400])
erddap_info(out$info$dataset_id[678])

out <- erddap_info(datasetid='noaa_ngdc_34bf_a95c_7e28')
## See brief overview of the variables and range of possible values, if given
out$variables
## all information on longitude
out$alldata$longitude
## all information on Climatological_Temperature
out$alldata$Climatological_Temperature

# table dap datasets
(out <- erddap_search(query='temperature', which = "table"))
erddap_info(out$info$dataset_id[1])
erddap_info(out$info$dataset_id[2])
erddap_info(out$info$dataset_id[3])
erddap_info(out$info$dataset_id[4])
erddap_info(out$info$dataset_id[54])
erddap_info(datasetid='erdCalCOFIshsiz')
out <- erddap_info(datasetid='erdCinpKfBT')
## 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

## End(Not run)
Search for ERDDAP tabledep or griddap datasets.

**Description**

Search for ERDDAP tabledep or griddap datasets.

**Usage**

```r
erddap_search(query, page = NULL, page_size = NULL, which = "griddap",
               ...)  
erddap_datasets(which = "tabledap")
```

**Arguments**

- `query` (character) Search terms
- `page` (integer) Page number
- `page_size` (integer) Results per page
- `which` (character) One of tabledep or griddap.
- `...` Further args passed on to `GET` (must be a named parameter)

**Author(s)**

Scott Chamberlain <myrmecocystus@gmail.com>

**References**


**Examples**

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

# List datasets
head( erddap_datasets('table') )
head( erddap_datasets('grid') )

## End(Not run)
```
erddap_table

Get ERDDAP tabledap data.

Description
Get ERDDAP tabledap data.

Usage
erddap_table(x, ..., fields = NULL, distinct = FALSE, orderby = NULL,
orderbymax = NULL, orderbymin = NULL, orderbyminmax = NULL,
units = NULL, store = disk(), callopts = list())

Arguments
x Anyting coercable to an object of class erddap_info. So the output of a call to
erddap_info, or a datasetid, which will internally be passed through erddap_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 vari-
able 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)).
store One of disk (default) or memory. You can pass options to disk. See Details.
callopts Further args passed on to GET
Details

Details:

Variables

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.

Where does the data go?

You can choose where data is stored. Be careful though. With griddap data, you can easily get a single file of hundreds of MB’s or GB’s in size with a single request. Using disk() caches files based on the URL of the request you perform, which is the combination of all parameters passed in, so if you refine a query by certain fields, etc., you will cach a different file. If you choose overwrite=TRUE within the disk() function then you’ll force writing a new file to disk. When you use memory(), no files are cached, data is stored in R’s memory.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>

References

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

See Also

erddap_grid erddap_clear_cache

Examples

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

# Pass time constraints
erddap_table('erdCalCOFIfshsiz', 'time>=2001-07-07', 'time<=2001-07-08')

# Pass in fields (i.e., columns to retrieve) & time constraints
erddap_table('erdCalCOFIfshsiz','time>=2001-07-07','time<=2001-07-10',
```
erddap_table

```
fields=c('longitude','latitude','fish_size','itis_tsn'))
erddap_table('erdCingKfmT', fields=c('latitude','longitude',
'Aplysia_californica_Mean_Density','Muricea_californica_Mean_Density'),
'time'>2007-06-24', 'time'<2007-07-01')

# Get info on a datasetid, then get data given information learned
erddap_info('erdCalCOF1rLrvsz')$variables
erddap_table('erdCalCOF1rLrvsz', fields=c('latitude','longitude','larvae_size',
'itis_tsn'), 'time'>2011-10-25', 'time'<2011-10-31')

# An example workflow
## Search for data
(out <- erddap_search(query='fish', which = 'table'))
## Using a datasetid, search for information on a datasetid
id <- out$info$dataset_id[7]
erddap_info(id)$variables
## Get data from the dataset
erddap_table(id, fields = c('fish','landings','year'))

# Time constraint
## Limit by time with date only
(info <- erddap_info('erdCalCOF1fshsz'))
erddap_table(info, fields = c('latitude','longitude','scientific_name'),
'time'>2001-07-14')

# Use distinct parameter
erddap_table('erdCalCOF1fshsz',fields=c('longitude','latitude','fish_size','itis_tsn'),
'time'>2001-07-07', 'time'<2001-07-10', distinct=TRUE)

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

# Use orderby parameter
erddap_table('erdCingKfmT', fields=c('longitude','latitude','time','temperature'),
'time'>2007-09-19', 'time'<2007-09-21', orderby='temperature')

# Use orderbymax parameter
erddap_table('erdCingKfmT', fields=c('longitude','latitude','time','temperature'),
'time'>2007-09-19', 'time'<2007-09-21', orderbymax='temperature')

# Use orderbymin parameter
erddap_table('erdCingKfmT', fields=c('longitude','latitude','time','temperature'),
'time'>2007-09-19', 'time'<2007-09-21', orderbymin='temperature')

# Use orderbyminmax parameter
erddap_table('erdCingKfmT', fields=c('longitude','latitude','time','temperature'),
'time'>2007-09-19', 'time'<2007-09-21', orderbyminmax='temperature')

# Use orderbymin parameter with multiple values
erddap_table('erdCingKfmT', fields=c('longitude','latitude','time','depth','temperature'),
'time'>2007-06-10', 'time'<2007-09-21', orderbymax=c('depth','temperature'))
```
Description

A dataset containing the FIPS codes for 51 US states and territories. The variables are as follows:

Format

A data frame with 3142 rows and 5 variables

Details

- state. US state name.
- county. County name.
- fips_state. Numeric value, from 1 to 51.
- fips_county. Numeric value, from 1 to 840.
- fips. Numeric value, from 1001 to 56045.
## Historical Observing Metadata Repository (HOMR) station metadata

### Description
Historical Observing Metadata Repository (HOMR) station metadata

### Usage

```sql
homr(qid = NULL, qidMod = NULL, station = NULL, state = NULL,
     county = NULL, country = NULL, name = NULL, nameMod = NULL,
     platform = NULL, date = NULL, begindate = NULL, enddate = NULL,
     headersOnly = FALSE, phrData = NULL, combine = FALSE, ...)
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qid</td>
<td>One of COOP, FAA, GHCND, ICAO, NCDCSTNID, NWSLI, TRANS, WBAN, or WMO, or any of those plus [a-zA-Z0-9], or just [a-zA-Z0-9]. (qid = qualified ID)</td>
</tr>
<tr>
<td>qidMod</td>
<td>(character) One of: is, starts, ends, contains. Specifies how the ID portion of the qid parameter should be applied within the search. If a qid is passed but the qidMod parameter is not used, qidMod is assumed to be IS.</td>
</tr>
<tr>
<td>station</td>
<td>(character) A station id.</td>
</tr>
<tr>
<td>state</td>
<td>(character) A two-letter state abbreviation. Two-letter code for US states, Canadian provinces, and other Island areas.</td>
</tr>
<tr>
<td>county</td>
<td>(character) A two letter county code. US county names, best used with a state identifier.</td>
</tr>
<tr>
<td>country</td>
<td>(character) A two letter country code. See here for a list of valid country names.</td>
</tr>
<tr>
<td>name</td>
<td>(character) One of name=[0-9A-Z]+. Searches on any type of name we have for the station.</td>
</tr>
<tr>
<td>nameMod</td>
<td>(character) [is</td>
</tr>
<tr>
<td>platform</td>
<td>(character) (aka network) ASOS</td>
</tr>
<tr>
<td>date</td>
<td>(character) [YYYY-MM-DD</td>
</tr>
<tr>
<td>begindate, enddate</td>
<td>[YYYY-MM-DD]. Limits values to only those that occurred within a date range.</td>
</tr>
<tr>
<td>headersOnly</td>
<td>(logical) Returns only minimal information for each station found (NCDC Station ID, Preferred Name, Station Begin Date, and Station End Date), but is much quicker than a full query. If you are performing a search that returns a large number of stations and intend to choose only one from that list to examine in detail,</td>
</tr>
</tbody>
</table>
headersOnly may give you enough information to find the NCDC Station ID for the station that you actually want.

**phrData** (logical) The HOMR web service now includes PHR (element-level) data when available, in an elements section. Because of how this data is structured, it can substantially increase the size of any result which includes it. If you don’t need this data you can omit it by including phrData=false. If the parameter is not set, it will default to phrData=true.

**combine** (logical) Combine station metadata or not.

... Further named parameters, such as query, path, etc, passed on to `modify_url`. Unnamed parameters will be combined with `config`.

**Details**

Since the definitions for variables are always the same, we don’t include the ability to get description data in this function. Use `link[rnoaa]{homr_descriptions}` to get descriptions information.

**Value**

A list, with elements named by the station ids.

**References**

http://www.ncdc.noaa.gov/homr/api

**Examples**

```r
## Not run:
homr(qid = 'COOP:046742')
homr(headersOnly=TRUE, qid='TRANS:')
homr(qid = ':046742')
homr(qid = 'FAA:')
homr(qidMod='starts', qid='COOP:0467')
homr(headersOnly=TRUE, state='DE')
homr(headersOnly=TRUE, country='GHANA')
homr(headersOnly=TRUE, state='NC', county='BUNCOMBE')
homr(name='CLAYTON')
res <- homr(state='NC', county='BUNCOMBE', combine=TRUE)
res$id
res$head
res$updates
homr(nameMod='starts', name='CLAY')
homr(headersOnly=TRUE, platform='ASOS')
homr(qid='COOP:046742', date='2011-01-01')
homr(qid='COOP:046742', beindate='2005-01-01', enddate='2011-01-01')
homr(state='DE', headersOnly=TRUE)
homr(station=20002078)
homr(station=20002078, date='all', phrData=FALSE)

## End(Not run)
```
Historical Observing Metadata Repository (HOMR) station metadata - definitions

Description
Historical Observing Metadata Repository (HOMR) station metadata - definitions

Usage
homr_definitions(...)

Arguments
... Named parameters, such as query, path, etc, passed on to modify_url. Unnamed parameters will be combined with config.

Examples
## Not run:
head(homr_definitions())

## End(Not run)

Get NOAA ISD/ISH data from NOAA FTP server.

Description
Get NOAA ISD/ISH data from NOAA FTP server.

Usage
isd(usaf = NULL, wban = NULL, year = NULL, path = "~/.rnoaa/isd", overwrite = TRUE)

isd_stations(...)

Arguments
usaf USAF code
wban WBAN code
year (numeric) One of the years from 1901 to the current year
path (character) A path to store the files, Default: ~/.rnoaa/isd
overwrite (logical) To overwrite the path to store files in or not, Default: TRUE.
... Curl options passed on to GET
Examples

## Not run:

```r
# Get station table
stations <- isd_stations()
head(stations)
```

```r
# Get data
(res <- isd(usaf="010230", wban="99999", year=1986))
(res <- isd(usaf="992230", wban="99999", year=1986))
```

## End(Not run)

---

**ncdc**

Search for and get NOAA NCDC data.

---

### Description

Search for and get NOAA NCDC data.

### Usage

```r
ncdc(datasetid = NULL, datatypeid = NULL, stationid = NULL,
      locationid = NULL, startdate = NULL, enddate = NULL, sortfield = NULL,
      sortorder = NULL, limit = 25, offset = NULL, token = NULL,
      dataset = NULL, datatype = NULL, station = NULL, location = NULL,
      locationtype = NULL, page = NULL, year = NULL, month = NULL,
      day = NULL, includemetadata = TRUE, results = NULL, ...)
```

### Arguments

- **datasetid** *(required)* Accepts a single valid dataset id. Data returned will be from the dataset specified, see datasets()
- **datatypeid** Accepts a valid data type id or a chain of data type ids in a comma-separated vector. Data returned will contain all of the data type(s) specified *(optional)*
- **stationid** Accepts a valid station id or a chain of station ids in a comma-separated vector. Data returned will contain data for the station(s) specified *(optional)*
- **locationid** Accepts a valid location id or a chain of location ids in a comma-separated vector. Data returned will contain data for the location(s) specified *(optional)*
- **startdate** *(required)* Accepts valid ISO formated date (yyyy-mm-dd) or date time (YYYY-MM-DDTh:mm:ss). Data returned will have data after the specified date. The date range must be less than 1 year.
- **enddate** *(required)* Accepts valid ISO formated date (yyyy-mm-dd) or date time (YYYY-MM-DDTh:mm:ss). Data returned will have data before the specified date. The date range must be less than 1 year.
- **sortfield** The field to sort results by. Supports id, name, mindate, maxdate, and datacoverage fields *(optional)*
sortorder  Which order to sort by, asc or desc. Defaults to asc (optional)
limit  Defaults to 25, limits the number of results in the response. Maximum is 1000 (optional)
offset  Defaults to 0, used to offset the resultlist (optional)
token  This must be a valid token token supplied to you by NCDC’s Climate Data Online access token generator. (required) Get an API key (=token) at http://www.ncdc.noaa.gov/cdo-web/token. You can pass your token in as an argument or store it in your .Rprofile file with an entry like
  • options("noaakey" = "your-noaa-token")
dataset  THIS IS A DEPRECATED ARGUMENT. See datasetid.
datatype  THIS IS A DEPRECATED ARGUMENT. See datatypeid.
station  THIS IS A DEPRECATED ARGUMENT. See stationid.
location  THIS IS A DEPRECATED ARGUMENT. See locationid.
locationtype  THIS IS A DEPRECATED ARGUMENT. There is no equivalent argument in v2 of the NOAA API.
page  THIS IS A DEPRECATED ARGUMENT. There is no equivalent argument in v2 of the NOAA API.
year  THIS IS A DEPRECATED ARGUMENT. Use combination of startdate and enddate arguments.
month  THIS IS A DEPRECATED ARGUMENT. Use combination of startdate and enddate arguments.
day  THIS IS A DEPRECATED ARGUMENT. Use combination of startdate and enddate arguments.
includemetadata  Used to improve response time by preventing the calculation of result metadata. Default: TRUE. This does not affect the return object, in that the named part of the output list called "meta" is still returned, but is NULL. In practice, I haven’t seen response time’s improve, but perhaps they will for you.
results  THIS IS A DEPRECATED ARGUMENT. See limit.
...  Further named parameters, such as query, path, etc, passed on to modify_url. Unnamed parameters will be combined with config.

Details

Note that NOAA NCDC API calls can take a long time depending on the call. The NOAA API doesn’t perform well with very long timespans, and will time out and make you angry - beware.
Keep in mind that three parameters, datasetid, startdate, and enddate are required.
Note that the default limit (no. records returned) is 25. Look at the metadata in $meta to see how many records were found. If more were found than 25, you could set the parameter limit to something higher than 25.
The attributes, or “flags”, for each row of the output for data may have a flag with it. Each datasetid has it’s own set of flags. The following are flag columns, and what they stand for. fl_ is the beginning of each flag column name, then one or more characters to describe the flag, keeping it
short to maintain a compact data frame. Some of these fields are the same across datasetids. See the vignette vignette("rnoaa_attributes", "rnoaa") for description of possible values for each flag.

- \texttt{fl\_c} completeness
- \texttt{fl\_d} day
- \texttt{fl\_m} measurement
- \texttt{fl\_q} quality
- \texttt{fl\_s} source
- \texttt{fl\_t} time
- \texttt{fl\_cmiss} consecutive missing
- \texttt{fl\_miss} missing
- \texttt{fl\_u} units

**Value**

An S3 list of length two, a slot of metadata (meta), and a slot for data (data). The meta slot is a list of metadata elements, and the data slot is a data.frame, possibly of length zero if no data is found.

**Examples**

```r
## Not run:
# GHCN-Daily (or GHCND) data, for a specific station
ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', startdate = '2013-10-01',
      enddate = '2013-12-01')

# GHCND data, for a location by FIPS code
ncdc(datasetid='GHCND', locationid = 'FIPS:02', startdate = '2010-05-01',
      enddate = '2010-05-10')

# GHCND data from October 1 2013 to December 1 2013
ncdc(datasetid='GHCND', startdate = '2013-10-01', enddate = '2013-10-05')

# GHCN-Monthly (or GHCNMS) data from October 1 2013 to December 1 2013
ncdc(datasetid='GHCNMS', startdate = '2013-10-01', enddate = '2013-12-01')

# Normals Daily (or NORMAL\_DLY) GHCND:USW00014895 dly-tmax-normal data
ncdc(datasetid='NORMAL\_DLY', stationid='GHCND:USW00014895', startdate = '2010-05-01',
      enddate = '2010-05-10')

# Dataset, and location in Australia
ncdc(datasetid='GHCND', locationid='FIPS:AS', startdate = '2010-05-01', enddate = '2010-05-31')

# Dataset, location and datatype for PRECIP\_HLY data
ncdc(datasetid='PRECIP\_HLY', locationid='ZIP:28801', datatypeid='HPCP',
     startdate = '2010-05-01', enddate = '2010-05-10')

# Dataset, location, station and datatype
ncdc(datasetid='PRECIP\_HLY', locationid='ZIP:28801', stationid='COOP:310301', datatypeid='HPCP',
     startdate = '2010-05-01', enddate = '2010-05-10')
```
ncdc_combine

startdate = '2010-05-01', enddate = '2010-05-10')

# Dataset, location, and datatype for GHCND
ncdc(datasetid='GHCND', locationid='FIPS:BR', datatypeid='PRCP', startdate = '2010-05-01',
     enddate = '2010-05-10')

# Normals Daily GHCND dly-tmax-normal data
ncdc(datasetid='NORMAL_DLY', datatypeid='dly-tmax-normal', startdate = '2010-05-01',
     enddate = '2010-05-10')

# Normals Daily GHCND:USW00014895 dly-tmax-normal
ncdc(datasetid='NORMAL_DLY', stationid='GHCND:USW00014895', datatypeid='dly-tmax-normal',
     startdate = '2010-05-01', enddate = '2010-05-10')

# Hourly Precipitation data for ZIP code 28801
ncdc(datasetid='PRECIP_HLY', locationid='ZIP:28801', datatypeid='HPCP',
     startdate = '2010-05-01', enddate = '2010-05-10')

# 15 min Precipitation data for ZIP code 28801
ncdc(datasetid='PRECIP_15', datatypeid='QPCP', startdate = '2010-05-01', enddate = '2010-05-02')

# Search the NORMAL_HLY dataset
ncdc(datasetid='NORMAL_HLY', stationid = 'GHCND:USW00003812', startdate = '2010-05-01',
     enddate = '2010-05-10')

# Search the ANNUAL dataset
ncdc(datasetid='ANNUAL', locationid='ZIP:28801', startdate = '2010-05-01',
     enddate = '2010-05-10')

# Search the NORMAL ANN dataset
ncdc(datasetid='NORMAL ANN', datatypeid='ANN-DUTR-NORMAL', startdate = '2010-01-01',
     enddate = '2010-01-01')

# Include metadata or not
ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', startdate = '2013-10-01',
     enddate = '2013-12-01')
nncdc(datasetid='GHCND', stationid='GHCND:USW00014895', startdate = '2013-10-01',
     enddate = '2013-12-01', includemetadata=FALSE)

## End(Not run)

## Not run:
# NEXRAD2 data
## doesn't work yet
ncdc(datasetid='NEXRAD2', startdate = '2013-10-01', enddate = '2013-12-01')

## End(Not run)

ncdc_combine

Coerce multiple outputs to a single data.frame object.
Description

Coerce multiple outputs to a single data.frame object.

Usage

ncdc_combine(...)

Arguments

... 

Objects from another ncdc_* function.

Value

A data.frame

Examples

### Not run:

# data
out1 <- ncdc(datasetid='GHND', locationid = 'FIPS:02', startdate = '2010-05-01',
enddate = '2010-05-31', limit=10)
out2 <- ncdc(datasetid='GHND', locationid = 'FIPS:02', startdate = '2010-07-01',
enddate = '2010-07-31', limit=10)
ncdc_combine(out1, out2)

# data sets
out1 <- ncdc_datasets(datatypeid='TOBS')
out2 <- ncdc_datasets(datatypeid='PRCP')
ncdc_combine(out1, out2)

# data types
out1 <- ncdc_datatypes(datatypeid='ACMH')
out2 <- ncdc_datatypes(datatypeid='PRCP')
ncdc_combine(out1, out2)

# data categories
out1 <- ncdc_datacats(datacategoryid="ANNAGR")
out2 <- ncdc_datacats(datacategoryid="PRCP")
ncdc_combine(out1, out2)

# data locations
out1 <- ncdc_locs(locationcategoryid='ST', limit=52)
out2 <- ncdc_locs(locationcategoryid='CITY', sortfield='name', sortorder='desc')
ncdc_combine(out1, out2)

# data locations
out1 <- ncdc_locs_cats(startdate='1970-01-01')
out2 <- ncdc_locs_cats(locationcategoryid='CLIM_REG')
ncdc_combine(out1, out2)

# stations
out1 <- ncdc_stations(datasetid='GHND', locationid='FIPS:12017',
ncdc_datacats

Get possible data categories for a particular datasetid, locationid, stationid, etc.

Description

Data Categories represent groupings of data types.

Usage

ncdc_datacats(datasetid = NULL, datacategoryid = NULL, stationid = NULL, locationid = NULL, startdate = NULL, enddate = NULL, sortfield = NULL, sortorder = NULL, limit = 25, offset = NULL, token = NULL, ...)  

Arguments

datasetid 
Accepts a single valid dataset id. Data returned will be from the dataset specified, see datasets() (required)

datacategoryid 
A valid data category id. Data types returned will be associated with the data category(ies) specified

stationid 
Accepts a valid station id. Data returned will contain data for the station(s) specified (optional)

locationid 
Accepts a valid location id. Data returned will contain data for the location(s) specified (optional)

startdate 
Accepts valid ISO formatted date (yyyy-mm-dd). Data returned will have data after the specified date. Parameter can be use independently of enddate (optional)

enddate 
Accepts valid ISO formatted date (yyyy-mm-dd). Data returned will have data before the specified date. Parameter can be use independently of startdate (optional)

sortfield 
The field to sort results by. Supports id, name, mindate, maxdate, and datacoverage fields (optional)
sortorder  Which order to sort by, asc or desc. Defaults to asc (optional)
limit  Defaults to 25, limits the number of results in the response. Maximum is 1000 (optional)
offset  Defaults to 0, used to offset the resultList (optional)
token  This must be a valid token token supplied to you by NCDC’s Climate Data Online access token generator. (required) Get an API key (=token) at http://www.ncdc.noaa.gov/cdo-web/token. You can pass your token in as an argument or store it in your .Rprofile file with an entry like
  * options("noaakey" = "your-noaa-token")
  ...
Further named parameters, such as query, path, etc, passed on to modify_url. Unnamed parameters will be combined with config.

Details
Note that calls with both startdate and enddate don’t seem to work, though specifying one or the other mostly works.

Value
A data.frame for all datasets, or a list of length two, each with a data.frame.

References
Vignette at http://ropensci.org/tutorials/rnoaa_tutorial.html

Examples
```r
## Not run:
## Limit to 10 results
ncdc_datacats(limit=10)

## Single data category
ncdc_datacats(datacategoryid="ANNAGR")

## Fetch data categories for a given set of locations
ncdc_datacats(locationid='CITY:US390029')
necd_datacats(locationid=c('CITY:US390029', 'FIPS:37'))

## Data categories for a given date
ncdc_datacats(startdate = '2013-10-01')

## Curl debugging
ncdc_datacats(limit=10, config=verbose())
out <- ncdc_datacats(limit=10, config=progress())

## End(Not run)
```
Description

From the NOAA API docs: All of our data are in datasets. To retrieve any data from us, you must know what dataset it is in.

Usage

```python
ncdc_datasets(datasetid = NULL, datatypeid = NULL, stationid = NULL, locationid = NULL, startdate = NULL, enddate = NULL, sortfield = NULL, sortorder = NULL, limit = 25, offset = NULL, token = NULL, dataset = NULL, page = NULL, year = NULL, month = NULL, ...)
```

Arguments

datasetid (optional) Accepts a single valid dataset id. Data returned will be from the dataset specified, see datasets()
datatypeid Accepts a valid data type id or a chain of data type ids in a comma-separated vector. Data returned will contain all of the data type(s) specified (optional)
stationid Accepts a valid station id or a chain of of station ids in a comma-separated vector. Data returned will contain data for the station(s) specified (optional)
locationid Accepts a valid location id or a chain of location ids in a comma-separated vector. Data returned will contain data for the location(s) specified (optional)
startdate (optional) Accepts valid ISO formated date (yyyy-mm-dd) or date time (YYYY-MM-DDThh:mm:ss). Data returned will have data after the specified date. The date range must be less than 1 year.
enddate (optional) Accepts valid ISO formated date (yyyy-mm-dd) or date time (YYYY-MM-DDThh:mm:ss). Data returned will have data before the specified date. The date range must be less than 1 year.
sortfield The field to sort results by. Supports id, name, mindate, maxdate, and datacoverage fields (optional)
sortorder Which order to sort by, asc or desc. Defaults to asc (optional)
limit Defaults to 25, limits the number of results in the response. Maximum is 1000 (optional)
offset Defaults to 0, used to offset the resultlist (optional)
token This must be a valid token token supplied to you by NCDC’s Climate Data Online access token generator. (required) Get an API key (=token) at http://www.ncdc.noaa.gov/cdo-web/token. You can pass your token in as an argument or store it in your .Rprofile file with an entry like
  ```r
  options("noaakey" = "your-noaa-token")
  ```
dataset THIS IS A DEPRECATED ARGUMENT. See datasetid.
ncdc_datatypes

page  THIS IS A DEPRECATED ARGUMENT. There is no equivalent argument in v2 of the NOAA API.

year  THIS IS A DEPRECATED ARGUMENT. Use combination of startdate and enddate arguments.

month  THIS IS A DEPRECATED ARGUMENT. Use combination of startdate and enddate arguments.

...  Further named parameters, such as query, path, etc, passed on to modify_url. Unnamed parameters will be combined with config.

Value

A data.frame for all datasets, or a list of length two, each with a data.frame.

Examples

```r
## Not run:
# Get a table of all datasets
ncdc_datasets()

# Get details from a particular dataset
ncdc_datasets(datasetid='ANNUAL')

# Get datasets with Temperature at the time of observation (TOBS) data type
ncdc_datasets(datatypeid='TOBS')

# Get datasets with data for a series of the same parameter arg, in this case stationid's
ncdc_datasets(stationid=c('COOP:310090','COOP:310184','COOP:310212'))

# Multiple datatypeid's
ncdc_datasets(datatypeid=c('ACMC','ACMH','ACSC'))
ncdc_datasets(datasetid='ANNUAL', datatypeid=c('ACMC','ACMH','ACSC'))

## End(Not run)
```

ncdc_datatypes  Get possible data types for a particular dataset

Description

From the NOAA API docs: Describes the type of data, acts as a label. If it’s 64 degrees out right now, then the data type is Air Temperature and the data is 64.

Usage

```r
ncdc_datatypes(datasetid = NULL, datatypeid = NULL, datacategoryid = NULL, 
stationid = NULL, locationid = NULL, startdate = NULL, enddate = NULL, 
sortfield = NULL, sortorder = NULL, limit = 25, offset = NULL, 
token = NULL, dataset = NULL, page = NULL, filter = NULL, ...)```
Arguments

datasetid (optional) Accepts a single valid dataset id. Data returned will be from the dataset specified, see datasets()
datatypeid Accepts a valid data type id or a chain of data type ids in a comma-separated vector. Data returned will contain all of the data type(s) specified (optional)
datacategoryid Optional. Accepts a valid data category id or a chain of data category ids separated by ampersands (although it is rare to have a data type with more than one data category). Data types returned will be associated with the data category(ies) specified
stationid Accepts a valid station id or a chain of station ids in a comma-separated vector. Data returned will contain data for the station(s) specified (optional)
locationid Accepts a valid location id or a chain of location ids in a comma-separated vector. Data returned will contain data for the location(s) specified (optional)
startdate (optional) Accepts valid ISO formatted date (yyyy-mm-dd) or date time (YYYY-MM-DDThh:mm:ss). Data returned will have data after the specified date. The date range must be less than 1 year.
enddate (optional) Accepts valid ISO formatted date (yyyy-mm-dd) or date time (YYYY-MM-DDThh:mm:ss). Data returned will have data before the specified date. The date range must be less than 1 year.
sortfield The field to sort results by. Supports id, name, mindate, maxdate, and datacoverage fields (optional)
sortorder Which order to sort by, asc or desc. Defaults to asc (optional)
limit Defaults to 25, limits the number of results in the response. Maximum is 1000 (optional)
offset Defaults to 0, used to offset the resultlist (optional)
token This must be a valid token token supplied to you by NCDC’s Climate Data Online access token generator. (required) Get an API key (=token) at http://www.ncdc.noaa.gov/cdo-web/token. You can pass your token in as an argument or store it in your .Rprofile file with an entry like
• options("noaakey" = "your-noaa-token")
dataset THIS IS A DEPRECATED ARGUMENT. See datasetid.
page THIS IS A DEPRECATED ARGUMENT. There is no equivalent argument in v2 of the NOAA API.
filter THIS IS A DEPRECATED ARGUMENT. There is no equivalent argument in v2 of the NOAA API.
...
Further named parameters, such as query, path, etc, passed on to modify_url. Unnamed parameters will be combined with config.

Value

A data.frame for all datasets, or a list of length two, each with a data.frame.
Examples

```r
## Not run:
# Fetch available data types
ncdc_datatypes()

# Fetch more information about the ACMH data type id
ncdc_datatypes(datatypeid="ACMH")

# Fetch data types with the air temperature data category
ncdc_datatypes(datacategoryid="TEMP", limit=56)

# Fetch data types that support a given set of stations
ncdc_datatypes(stationid=c('COOP:310090','COOP:310184','COOP:310212'))

## End(Not run)
```

ncdc_legacy

Search the NOAA NCDC legacy API - main purpose to get ISD daily data.

Description

Search the NOAA NCDC legacy API - main purpose to get ISD daily data.

Usage

```r
ncdc_leg_variables(dataset = "isd", variable = NULL, token = NULL, ...)
ncdc_leg_sites(dataset = "isd", state_id = NULL, state_abbrev = NULL,
  country_id = NULL, country_abbrev = NULL, token = NULL, ...)
ncdc_leg_site_info(dataset = "isd", station, token = NULL, ...)
ncdc_leg_data(dataset = "isd", station, variable, start_date, end_date,
  token = NULL, ...)
```

Arguments

dataset          Dataset id, one of isd (default), ish, or daily.
varyable         A variable id.
token            Your token. Store as environment variable as NOAA_LEG_TOKEN or option variable as noaalegtoken.
...               Further args passed to GET
state_id         State ID
state_abbrev     State abbreviation
country_id       Country id
ncdc_locs

Get metadata about NOAA NCDC locations.

Description

From the NOAA NCDC API docs: Locations can be a specific latitude/longitude point such as a station, or a label representing a bounding area such as a city.

country_abbrev  Country abbreviation
station         A station id.
start_date,end_date
          Date to start and end search for. Valid formats include, YYYY, YYYYMM, YYYYMMDD, YYYYMMDDhh, and YYYYMMDDhhmm.

Details

BEWARE: This service, as far as I can tell, enforces a wait time between successive data requests of 60 seconds. This is indeed a long time.

References


Examples

```r
## Not run:
# Variables
## All variables
ncdc_leg_variables()
# Specific variables
### Air Temperature Observation
ncdc_leg_variables(variable = "TMP")
### Snow Accumulation For The Month
ncdc_leg_variables(variable = "AN1")

# Sites
head(ncdc_leg_sites())
ncdc_leg_sites(country_id = 1)
ncdc_leg_sites(country_abbrev = 'CA')
ncdc_leg_sites(state_id = 11)
ncdc_leg_sites(state_abbrev = 'HI')
## site info
ncdc_leg_site_info(stationid=71238099999)

# Data
ncdc_leg_data('isd', 71238099999, 'TMP', 200101010000, 200101312359)
ncdc_leg_data('isd', 71238099999, 'TMP', 200101, 200201)

## End(Not run)
```
Usage

ncdc_locs(datasetid = NULL, locationid = NULL, locationcategoryid = NULL,
           startdate = NULL, enddate = NULL, sortfield = NULL, sortorder = NULL,
           limit = 25, offset = NULL, token = NULL, ...)

Arguments

datasetid      A single valid dataset id. Data returned will be from the dataset specified, see datasets() (required)
locationid     A valid location id or a chain of location ids seperated by ampersands. Data returned will contain data for the location(s) specified (optional)
locationcategoryid
               A valid location id or a chain of location category ids in a comma-separated vector. Locations returned will be in the location category(ies) specified
startdate      A valid ISO formated date (yyyy-mm-dd). Data returned will have data after the specified date. Paramater can be use independently of enddate (optional)
enddate        Accepts valid ISO formated date (yyyy-mm-dd). Data returned will have data before the specified date. Parameter can be use independently of startdate (optional)
sortfield     The field to sort results by. Supports id, name, mindate, maxdate, and datacoverage fields (optional)
sortorder      Which order to sort by, asc or desc. Defaults to asc (optional)
limit          Defaults to 25, limits the number of results in the response. Maximum is 1000 (optional)
offset         Defaults to 0, used to offset the resultlist (optional)
token          This must be a valid token token supplied to you by NCDC’s Climate Data Online access token generator. (required) Get an API key (=token) at http://www.ncdc.noaa.gov/cdo-web/token. You can pass your token in as an argument or store it in your .Rprofile file with an entry like
                 • options("noakey" = "your-noaa-token")
                 ...
Value

A list containing metadata and the data, or a single data.frame.

Examples

## Not run:
# All locations, first 25 results
ncdc_locs()

# Fetch more information about location id FIPS:37
ncdc_locs(locationid='FIPS:37')
# Fetch available locations for the GHCND (Daily Summaries) dataset
ncdc_locs(datasetid='GHCND')

# Fetch all U.S. States
ncdc_locs(locationcategoryid='ST', limit=52)

# Fetch list of city locations in descending order
ncdc_locs(locationcategoryid='CITY', sortfield='name', sortorder='desc')

## End(Not run)

ncdc_locs_cats

Get metadata about NOAA location categories.

### Description
Location categories are groupings of similar locations.

### Usage
ncdc_locs_cats(datasetid = NULL, locationcategoryid = NULL,
startdate = NULL, enddate = NULL, sortfield = NULL, sortorder = NULL,
limit = 25, offset = NULL, token = NULL, ...)

### Arguments
- **datasetid**
  A single valid dataset id. Data returned will be from the dataset specified, see datasets() (required)
- **locationcategoryid**
  A valid location id or a chain of location category ids in a comma-separated vector. Locations returned will be in the location category(ies) specified
- **startdate**
  A valid ISO formatted date (yyyy-mm-dd). Data returned will have data after the specified date. Parameter can be use independently of enddate (optional)
- **enddate**
  Accepts valid ISO formatted date (yyyy-mm-dd). Data returned will have data before the specified date. Parameter can be use independently of startdate (optional)
- **sortfield**
  The field to sort results by. Supports id, name, mindate, maxdate, and datacoverage fields (optional)
- **sortorder**
  Which order to sort by, asc or desc. Defaults to asc (optional)
- **limit**
  Defaults to 25, limits the number of results in the response. Maximum is 1000 (optional)
- **offset**
  Defaults to 0, used to offset the resultlist (optional)
- **token**
  This must be a valid token token supplied to you by NCDC’s Climate Data Online access token generator. (required) Get an API key (=token) at http://www.ncdc.noaa.gov/cdo-web/token. You can pass your token in as an argument or store it in your .Rprofile file with an entry like
options("noaakey" = "your-noaa-token")

Further named parameters, such as query, path, etc, passed on to `modify_url`. Unnamed parameters will be combined with `config`.

**Details**

Locations can be a specific latitude/longitude point such as a station, or a label representing a bounding area such as a city.

**Value**

A list containing metadata and the data, or a single data.frame.

**Examples**

```r
## Not run:
# All location categories, first 25 results
ncdc_locs_cats()

# Find locations with category id of CLIM_REG
ncdc_locs_cats(locationcategoryid="CLIM_REG")

# Displays available location categories within GHCN-Daily dataset
ncdc_locs_cats(datasetid="GHCND")

# Displays available location categories from start date 1970-01-01
ncdc_locs_cats(startdate='1970-01-01')

## End(Not run)
```

**ncdc_plot**

*Plot NOAA climate data.*

**Description**

This function accepts directly output from the `ncdc` function, not other functions.

**Usage**

```r
ncdc_plot(..., breaks = "7 days", dateformat = "%d/%m/%y")
```

```r
## S3 method for class 'ncdc_data'
ncdc_plot(..., breaks = "7 days",
          dateformat = "%d/%m/%y")
```

**Arguments**

- `...` Input noaa object or objects.
- `breaks` Regularly spaced date breaks for x-axis. See `date_breaks`
- `dateformat` Date format using standard POSIX specification for labels on x-axis. See `date_format`
Details

This is a simple wrapper function around some ggplot2 code. There is indeed a lot you can modify in your plots, so this function just does some basic stuff. Here’s the code within this function, where input is the output from a `ncdc` call - go crazy:

```r
input <- input$data
input$date <- ymd(str_replace(as.character(input$date), "T00:00:00.000", ")

ggplot(input, aes(date, value)) + theme_bw(base_size=18) + geom_line(size=2) + scale_x_datetime(breaks = date_breaks("7 days"), labels = date_format("labs(y=as.character(input[1,"dataType"]), x="Date")
```

Value

Plot of climate data.

Examples

```r
## Not run:
# Search for data first, then plot
out <- ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', datatimeid='PRCP',
startdate = '2010-05-01', enddate = '2010-10-31', limit=500)
ncdc_plot(out)
ncdc_plot(out, breaks="14 days")
nCDC_plot(out, breaks="1 month", dateformat="%d/%m")
nCDC_plot(out, breaks="1 month", dateformat="%d/%m")

out2 <- ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', datatimeid='PRCP',
startdate = '2010-05-01', enddate = '2010-05-03', limit=100)
nCDC_plot(out2, breaks="6 hours", dateformat="%H")

# Combine many calls to ncdc function
out1 <- ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', datatimeid='PRCP',
startdate = '2010-03-01', enddate = '2010-05-01', limit=500)
out2 <- ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', datatimeid='PRCP',
startdate = '2010-09-01', enddate = '2010-10-31', limit=500)
df <- ncdc_combine(out1, out2)
nCDC_plot(df)
## or pass in each element separately
ncdc_plot(out1, out2, breaks="45 days")

## End(Not run)
```

Description

From the NOAA NCDC API docs: Stations are where the data comes from (for most datasets) and can be considered the smallest granual of location data. If you know what station you want, you can quickly get all manner of data from it.
Usage

ncdc_stations(stationid = NULL, datasetid = NULL, datatypeid = NULL, locationid = NULL, startdate = NULL, enddate = NULL, sortfield = NULL, sortorder = NULL, limit = 25, offset = NULL, datacategoryid = NULL, extent = NULL, radius = 10, token = NULL, dataset = NULL, station = NULL, location = NULL, locationtype = NULL, page = NULL, ...)

Arguments

stationid  Accepts a valid station id or a chain of of station ids in a comma-separated vector. Data returned will contain data for the station(s) specified (optional)
datasetid  (optional) Accepts a single valid dataset id. Data returned will be from the dataset specified, see datasets()
datatypeid  Accepts a valid data type id or a chain of data type ids in a comma-separated vector. Data returned will contain all of the data type(s) specified (optional)
locationid  Accepts a valid location id or a chain of location ids in a comma-separated vector. Data returned will contain data for the location(s) specified (optional)
startdate  (optional) Accepts valid ISO formated date (yyyy-mm-dd) or date time (YYYY-MM-DDThh:mm:ss). Data returned will have data after the specified date. The date range must be less than 1 year.
enddate  (optional) Accepts valid ISO formated date (yyyy-mm-dd) or date time (YYYY-MM-DDThh:mm:ss). Data returned will have data before the specified date. The date range must be less than 1 year.
sortfield  The field to sort results by. Supports id, name, mindate, maxdate, and datacoverage fields (optional)
sortorder  Which order to sort by, asc or desc. Defaults to asc (optional)
limit  Defaults to 25, limits the number of results in the response. Maximum is 1000 (optional)
offset  Defaults to 0, used to offset the resultlist (optional)
datacategoryid  (character, optional) Accepts a valid data category id or an array of data category ids. Stations returned will be associated with the data category(ies) specified
extent  (numeric, optional) The geographical extent for which you want to search. Give either a vector with two values: a latitude and a longitude. For example, c(lat, long). Or give four values that defines a bounding box, lat and long for the southwest corner, then lat and long for the northeast corner. For example: c(minlat, minlong, maxlat, maxlong).
radius  (numeric) If a single latitude/longitude pair is given to the extent parameter, the radius to create around the point. Ignored if a vector of appropriate structure is passed to the extent parameter.
token  This must be a valid token token supplied to you by NCDC’s Climate Data Online access token generator. (required) Get an API key (=token) at http://www.ncdc.noaa.gov/cdo-web/token. You can pass your token in as an argument or store it in your .Rprofile file with an entry like
**ncdc_stations**

- `options("noaakey" = "your-noaa-token")`

**dataset**

This is a deprecated argument. See `datasetid`.

**station**

This is a deprecated argument. See `stationid`.

**location**

This is a deprecated argument. See `locationid`.

**locationtype**

This is a deprecated argument. There is no equivalent argument in v2 of the NOAA API.

**page**

This is a deprecated argument. There is no equivalent argument in v2 of the NOAA API.

... Further named parameters, such as `query`, `path`, etc, passed on to `modify_url`.

Unnamed parameters will be combined with `config`.

**Value**

A list of metadata.

**Examples**

```r
## Not run:
# Get metadata on all stations
cdc_stations()
cdc_stations(limit=5)

# Get metadata on a single station
cdc_stations(stationid='COOP:010008')

# Displays all stations within GHCN-Daily (100 Stations per page limit)
cdc_stations(datasetid='GHICND')

# Station
cdc_stations(datasetid='NORMAL_DLY', stationid='GHICND:USW00014895')

# Displays all stations within GHCN-Daily (Displaying page 10 of the results)
cdc_stations(datasetid='GHICND')

# Specify datasetid and locationid
cdc_stations(datasetid='GHICND', locationid='FIPS:12017')

# Specify datasetid, locationid, and station
cdc_stations(datasetid='GHICND', locationid='FIPS:12017', stationid='GHICND:USC00084289')

# Specify datasetid, locationidtype, locationid, and station
cdc_stations(datasetid='GHICND', locationid='FIPS:12017', stationid='GHICND:USC00084289')

# Displays list of stations within the specified county
cdc_stations(datasetid='GHICND', locationid='FIPS:12017')

# Displays list of Hourly Precipitation locationids between 01/01/1990 and 12/31/1990
cdc_stations(datasetid='PRECIP_HLY', startdate='19900101', enddate='19901231')

# Search for stations by spatial extent
```
## Search using a single point, given by a lat long pair
```r
cncd_stations(extent=c(33.95, -118.40))
```

## Search using a bounding box, w/ lat/long of the SW corner, then of NE corner
```r
cncd_stations(extent=c(47.5204, -122.2047, 47.6139, -122.1065))
```

## End (Not run)

---

### rnoaa-defunct

**Defunct functions in rnoaa**

**Description**

- **noaa**: Function name changed, prefixed with ncdc now
- **noaa_datacats**: Function name changed, prefixed with ncdc now
- **noaa_datasets**: Function name changed, prefixed with ncdc now
- **noaa_datatypes**: Function name changed, prefixed with ncdc now
- **noaa_locs**: Function name changed, prefixed with ncdc now
- **noaa_locs_cats**: Function name changed, prefixed with ncdc now
- **noaa_stations**: Function name changed, prefixed with ncdc now
- **noaa_plot**: Function name changed, prefixed with ncdc now
- **noaa_combine**: Function name changed, prefixed with ncdc now
- **noaa_seaice**: Function name changed to seaice
- **erddap_data**: See the function `erddap_grid` for getting griddap data and `erddap_table` for getting tabledap data.

---

### seaice

**Get sea ice data.**

**Description**

Get sea ice data.

**Usage**

```r
seaice(url, ...)
```

**Arguments**

- **url**: A url for a NOAA sea ice ftp file
- **...**: Further arguments passed on to readshpfile function, see `readshpfile`
storm_columns

Value

A data.frame

Examples

```r
## Not run:
# Look at data.frame's for a series of years for Feb, South pole
urls <- sapply(seq(1979,1990), function(x) seaiceeurls(yr=x, mo='Feb', pole='S'))
out <- lapply(urls, seaice)
lapply(out, head)

# Map a single year/month/pole combo
urls <- seaiceeurls(mo='Apr', pole='N', yr=1990)
out <- seaice(urls)
library('ggplotz')
ggplot(out, aes(long, lat, group=group)) +
  geom_polygon(fill="steelblue") +
  theme_ice()

# Map all years for April only for North pole
library('plyr')
library('doMC')
urls <- seaiceeurls(mo='Apr', pole='N')
registerDoMC(cores=4)
out <- ldply(urls, seaice, .parallel=TRUE)
names(out) <- seq(1979,2013,1)
df <- ldply(out)
ggplot(df, aes(long, lat, group=group)) +
  geom_polygon(fill="steelblue") +
  theme_ice() +
  facet_wrap(~ .id)

## End(Not run)
```

---

storm_columns

**NOAA storm column descriptions for data from IBTrACS**

Description

This dataset includes description of the columns of each dataset acquired using `storm_data`

Format

A data frame with 195 rows and 8 variables
storm_names  

**NOAA storm names from IBTrACS**

**Description**

This dataset includes a crosswalk from storm serial numbers to their names. Storm serial numbers are used to search for storms in the `storm_data` function.

**Format**

A data frame with 12,209 rows and 2 variables

---

storm_shp  

**Get NOAA wind storm tabular data, metadata, or shp files from IBTrACS**

**Description**

Get NOAA wind storm tabular data, metadata, or shp files from IBTrACS

**Usage**

```r
storm_shp(basin = NULL, storm = NULL, year = NULL, type = "points",
          path = "~/rnoaa/storms", overwrite = TRUE)

storm_shp_read(x)

storm_data(basin = NULL, storm = NULL, year = NULL,
           path = "~/rnoaa/storms", overwrite = TRUE)

storm_meta(what = "storm_columns")
```

**Arguments**

- **basin** (character) A basin name, one of EP, NA, NI, SA, SI, SP, or WP.
- **storm** (character) A storm serial number of the form YYYYJJJHTTNNN. See Details.
- **year** (numeric) One of the years from 1842 to 2014
- **type** (character) One of points or lines. This gives shp files with points, or with lines.
- **path** (character) A path to store the files, Default: `~/rnoaa/storms`
- **overwrite** (logical) To overwrite the path to store files in or not, Default: TRUE.
- **x** Output from `storm_shp`, a path to shp file to read in.
- **what** (character) One of `storm_columns` or `storm_names`. 


Details

International Best Track Archive for Climate Stewardship (IBTrACS)

Details for storm serial numbers:

- YYYY is the corresponding year of the first recorded observation of the storm
- JJJ is the day of year of the first recorded observation of the storm
- H is the hemisphere of the storm: N=Northern, S=Southern
- TT is the absolute value of the rounded latitude of the first recorded observation of the storm (range 0-90, if basin=SA or SH, then TT in reality is negative)
- NNN is the rounded longitude of the first recorded observation of the storm (range 0-359)

For example: 1970143N19091 is a storm in the North Atlantic which started on May 23, 1970 near 19°N 91°E


The datasets included in the package storm_names, and storm_columns may help in using these storm functions.

References


Examples

```r
## Not run:
# Metadata
head( storm_meta() )
head( storm_meta("storm_columns") )
head( storm_meta("storm_names") )

# Tabular data
## Get tabular data for basins, storms, or years
storm_data(basin='WP')
storm_data(storm='1970143N19091')
storm_data(year=1940)
storm_data(year=1941)
storm_data(year=2010)

## Or get all data, simply don't specify a value for basin, storm, or year
res <- storm_data(read=FALSE) # just get path
head()

# shp files
## storm_shp downloads data and gives a path back
## to read in, use storm_shp_read
res <- storm_shp(basin='EP')
storm_shp_read(res)

## Get shp file for a storm
(res2 <- storm_shp(storm='1970143N19091'))
```
## Plot shp file data, we'll need sp library
library('sp')

### for year 1940, points
(res3 <- storm_shp(year=1940))
res3shp <- storm_shp_read(res3)
plot(res3shp)

### for year 1940, lines
(res3_lines <- storm_shp(year=1940, type="lines"))
res3_linesshp <- storm_shp_read(res3_lines)
plot(res3_linesshp)

### for year 2010, points
(res4 <- storm_shp(year=2010))
res4shp <- storm_shp_read(res4)
plot(res4shp)

## End(Not run)

---

**swdi** 
*Get NOAA data for the severe weather data inventory (swdi).*

### Description
Get NOAA data for the severe weather data inventory (swdi).

### Usage
```r
swdi(dataset = NULL, format = "xml", startdate = NULL, enddate = NULL,
limit = 25, offset = NULL, radius = NULL, center = NULL,
bbox = NULL, tile = NULL, stat = NULL, id = NULL, filepath = NULL,
callopts = list())
```

### Arguments
- **dataset**: Dataset to query. See below for details.
- **format**: File format to download. One of xml, csv, shp, or kmz.
- **startdate**: Start date. See details.
- **enddate**: End date. See details.
- **limit**: Number of results to return. Defaults to 25. Any number from 1 to 10000000.
- **offset**: Any number from 1 to 10000000. Default is NULL, no offset, start from 1.
- **radius**: Search radius in miles (current limit is 15 miles)
- **center**: Center coordinate in lon, lat decimal degree format, e.g.: c(-95.45,36.88)
- **bbox**: Bounding box in format of minLon,minLat,maxLon,maxLat, e.g.: c(-91,30,-90,31)
tile Coordinate in lon,lat decimal degree format, e.g.: c(-95.45,36.88) The lat/lon values are rounded to the nearest tenth of degree. For the above example, the matching tile would contain values from -95.4500 to -95.5499 and 36.8500 to 36.9499

stat One of count or tilesum:$longitude,$latitude. Setting stat='count' returns number of results only (no actual data). stat='tilesum:$longitude,$latitude' returns daily feature counts for a tenth of a degree grid centered at the nearest tenth of a degree to the supplied values.

id An identifier, e.g., 533623. Not sure how you find these ids?

filepath If kmz or shp chosen the file name and optionally path to write to. Ignored

callopts Further arguments passed on to the API GET call. (optional)

Details
Options for the dataset parameter. One of (and their data formats):

- nx3tvs NEXRAD Level-3 Tornado Vortex Signatures (point)
- nx3meso NEXRAD Level-3 Mesocyclone Signatures (point)
- nx3hail NEXRAD Level-3 Hail Signatures (point)
- nx3structure NEXRAD Level-3 Storm Cell Structure Information (point)
- plsr Preliminary Local Storm Reports (point)
- warn Severe Thunderstorm, Tornado, Flash Flood and Special Marine warnings (polygon)
- nldn Lightning strikes from Vaisala (.gov and .mil ONLY) (point)

For startdate and enddate, the date range syntax is 'startDate:endDate' or special option of 'periodOfRecord'. Note that startDate is inclusive and endDate is exclusive. All dates and times are in GMT. The current limit of the date range size is one year.

All latitude and longitude values for input parameters and output data are in the WGS84 datum.

Value

If xml or csv chosen, a list of length three, a slot of metadata (meta), a slot for data (data), and a slot for shape file data with a single column 'shape'. The meta slot is a list of metadata elements, and the data slot is a data.frame, possibly of length zero if no data is found.

If kmz or shp chosen, the file is downloaded to your machine and a message is printed.

Examples

```r
# Not run:
# Search for nx3tvs data from 5 May 2006 to 6 May 2006
swdi(dataset='nx3tvs', startdate='20060505', enddate='20060506')

# Get all 'nx3tvs' within 15 miles of latitude = 32.7 and longitude = -102.0
swdi(dataset='nx3tvs', startdate='20060506', enddate='20060507',
radius=15, center=c(-102.0,32.7))
```
Get NOAA tornado data.

**Description**

Get NOAA tornado data.
Usage

tornadoes(path = "~/.rnoaa/tornadoes", overwrite = TRUE)

Arguments

path A path to store the files, Default: ~/.ots/kelp
overwrite (logical) To overwrite the path to store files in or not, Default: TRUE.

Value

A Spatial object is returned of class SpatialLinesDataFrame.

References

http://www.spc.noaa.gov/gis/svrgis/

Examples

## Not run:
shp <- tornadoes()
library('sp')
plot(shp) # may take 10 sec or so to render

## End(Not run)
Index

*Topic datasets
  fipscodes, 14
  storm_columns, 37
  storm_names, 38

*Topic package
  rnoaaMpackage, 2

as.erddap_info (erddap_info), 8
config, 16, 17, 19, 24, 26, 27, 30, 32, 35
date_breaks, 32
date_format, 32
disk, 3

erddap, 4
erddap_clear_cache, 5, 6, 12
erddap_data, 36
erddap_datasets, 4
erddap_datasets (erddap_search), 10
erddap_grid, 4, 5, 12, 36
erddap_info, 4, 8
erddap_search, 4, 10
erddap_table, 4, 6, 11, 36

fipscodes, 14
GET, 5, 8, 10, 11, 17, 28

homr, 15
homr_definitions, 17

isd, 17
isd_stations (isd), 17

memory (disk), 3
modify_url, 16, 17, 19, 24, 26, 27, 30, 32, 35

ncdc, 18, 32, 33
ncdc_combine, 21
ncdc_datacats, 23
ncdc_datasets, 25
ncdc_datatypes, 26
ncdc_leg_data (ncdc_legacy), 28
ncdc_leg_site_info (ncdc_legacy), 28
ncdc_leg_sites (ncdc_legacy), 28
ncdc_leg_variables (ncdc_legacy), 28
ncdc_legacy, 28
ncdc_locs, 29
ncdc_locs_cats, 31
ncdc_plot, 32
ncdc_stations, 33
noaa, 36
noaa_combine, 36
noaa_datacats, 36
noaa_datasets, 36
noaa_datatypes, 36
noaa_locs, 36
noaa_locs_cats, 36
noaa_plot, 36
noaa_seaice, 36
noaa_stations, 36

rnoaa (rnoaa-package), 2
rnoaa-defunct, 36
rnoaa-package, 2

seaice, 36
storm_columns, 37, 39
storm_data, 37, 38
storm_data (storm_shp), 38
storm_meta (storm_shp), 38
storm_names, 38, 39
storm_shp, 38
storm_shp_read (storm_shp), 38
swdi, 40

tornadoes, 42