Package 'rnoaa'

January 7, 2021

Title 'NOAA' Weather Data from R

Description Client for many 'NOAA' data sources including the 'NCDC' climate 'API' at <https://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, 'NOAA' Historical Observing 'Metadata' Repository ('HOMR') data, 'NOAA' storm data via 'IBTrACS', tornado data via the 'NOAA' storm prediction center, and more.

Version 1.3.0

License MIT + file LICENSE

Encoding UTF-8

Language en-US

URL https://docs.ropensci.org/rnoaa/ (docs), https://github.com/ropensci/rnoaa (devel)

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

LazyData true

Imports utils, crul (>= 0.7.0), lubridate, dplyr, tidyr, tidyselect, ggplot2, scales, XML, xml2, jsonlite, rappdirs, gridExtra, tibble, isdparser (>= 0.2.0), geonames, hoardr (>= 0.5.2), data.table

Suggests roxygen2 (>= 7.1.0), testthat, knitr, taxize, ncdf4, raster, leaflet, rgdal, rmarkdown, purrr, ggmap, ropenaq, vcr (>= 0.5.4), webmockr

RoxygenNote 7.1.1

X-schema.org-applicationCategory Climate

X-schema.org-keywords earth, science, climate, precipitation, temperature, storm, buoy, NCDC, NOAA, tornadoe, sea ice, ISD

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

NeedsCompilation no

Author Scott Chamberlain [aut, cre] (<https://orcid.org/0000-0003-1444-9135>),
Brooke Anderson [ctb],
Maëlle Salmon [ctb],
Adam Erickson [ctb],
Nicholas Potter [ctb],
Joseph Stachelek [ctb],
Alex Simmons [ctb],
Karthik Ram [ctb],
Hart Edmund [ctb],
rOpenSci [fnd] (https://ropensci.org)

Maintainer Scott Chamberlain <myrmecocystus@gmail.com>

Repository CRAN

Date/Publication 2021-01-07 22:10:05 UTC

R topics documented:

`rnoaa-package` *rnoaa*

Description

rnoaa is an R interface to NOAA climate data. For official package documentation see the "docs" link in DESCRIPTION

Data Sources

Many functions in this package interact with the National Climatic Data Center application programming interface (API) at 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 us. 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 from the National Buoy Data Center
- ghcnd_*/meteo_* - GHCND daily data from NOAA
- isd_* - ISD/ISH data from NOAA
- homr_* - Historical Observing Metadata Repository (HOMR) vignette
- ncdc_* - NOAA National Climatic Data Center (NCDC) vignette (examples)
- `sea_ice` - Sea ice vignette
- `storm_` - Storms (IBTrACS) vignette

- swdi - Severe Weather Data Inventory (SWDI) vignette
- tornadoes - From the NOAA Storm Prediction Center
- argo_* - Argo buoys
- coops_search - NOAA CO-OPS - tides and currents data
- cpc_prcp - rainfall data from the NOAA Climate Prediction Center (CPC)
- arc2 - rainfall data from Africa Rainfall Climatology version 2
- bsw - Blended sea winds (BSW)
- ersst - NOAA Extended Reconstructed Sea Surface Temperature (ERSST) data
- lcd - Local Climitalogical Data from NOAA

Where data comes from and government shutdowns

Government shutdowns can greatly affect data sources in this package. The following is a breakdown of the functions that fetch data by HTTP vs. FTP - done this way as we've noticed that during the ealry 2019 border wall shutdown most FTP services were up, while those that were down were HTTP; though not all HTTP services were down.

- HTTP info:
- FTP info:

HTTP services (whether service is/was up or down during early 2019 shutdown)

- buoy_* - Up
- homr_* - Up
- ncdc_* - Down
- swdi - Down
- tornadoes - Down
- argo_* - Up (all HTTP except two fxns, see also FTP below)
- coops_search - Up
- ersst - Down
- lcd - Down
- se_* - Down

FTP services (whether service is/was up or down during early 2019 shutdown)

- ghcnd_* - Up
- isd_* - Up
- sea_ice - Up
- storm_ - Up
- argo_* - Up (only two fxns: argo(), argo_buoy_files())
- cpc_prcp - Up
- arc2 - Up
- bsw - Up

We've tried to whenever possible detect whether a service is error due to a government shutdown and give a message saying so. If you know a service is down that rnoaa interacts with but we don't fail well during a shutdown let us know.

A note about NCDF data

Some functions use netcdf files, including:

- `ersst`
- `buoy`
- `bsw`
- `argo`

You'll need the `ncdf4` package for those functions, and those only. `ncdf4` is in Suggests in this package, meaning you only need `ncdf4` if you are using any of the functions listed above. You'll get an informative error telling you to install `ncdf4` if you don't have it and you try to use the those functions. Installation of `ncdf4` should be straightforward on any system.

The `meteo` family of functions

The `meteo` family of functions are prefixed with `meteo_` and provide a set of helper functions to:

- Identify candidate stations from a latitude/longitude pair
- Retrieve complete data for one or more stations (`meteo_coverage()`)

`arc2` *Arc2 - Africa Rainfall Climatology version 2*

Description

Arc2 - Africa Rainfall Climatology version 2

Usage

```
arc2(date, box = NULL, ...)
```

Arguments

`date`	(character/date) one or more dates of the form YYYY-MM-DD
`box`	(numeric) vector of length 4, of the form xmin, ymin, xmax, ymax. optional. If not given, no spatial filtering is done. If given, we use `dplyr::filter()` on a combined set of all dates, then split the output into tibbles by date
`...`	curl options passed on to crul::verb-GET

Value

a list of tibbles with columns:

- date: date (YYYY-MM-DD)
- lon: longitude
- lat: latitude
- precip: precipitation (mm)

box parameter

The box parameter filters the arc2 data to a bounding box you supply. The function that does the cropping to the bounding box is `dplyr::filter`. You can do any filtering you want on your own if you do not supply box and then use whatever tools you want to filter the data by lat/lon, date, precip values.

Note

See arc2_cache for managing cached files

References

docs: ftp://ftp.cpc.ncep.noaa.gov/fews/fewsdata/africa/arc2/ARC2_readme.txt

Examples

```
## Not run:
x = arc2(date = "1983-01-01")
arc2(date = "2017-02-14")

# many dates
arc2(date = c("2019-05-27", "2019-05-28"))
arc2(seq(as.Date("2019-04-21"), by = "day", length.out = 5))
## combine outputs
x <- arc2(seq(as.Date("2019-05-20"), as.Date("2019-05-25"), "days"))
dplyr::bind_rows(x)

# bounding box filter
box <- c(xmin = 9, ymin = 4, xmax = 10, ymax = 5)
arc2(date = "2017-02-14", box = box)
arc2(date = c("2019-05-27", "2019-05-28"), box = box)
arc2(seq(as.Date("2019-05-20"), as.Date("2019-05-25"), "days"), box = box)

## End(Not run)
```

---

argo *Get Argo buoy data*

Description

Get Argo buoy data

Usage

```
argo_search(
  func = NULL,
  of = NULL,
  qwmo = NULL,
```

```
  wmo = NULL,
  box = NULL,
  area = NULL,
  around = NULL,
  year = NULL,
  yearmin = NULL,
  yearmax = NULL,
  month = NULL,
  monthmin = NULL,
  monthmax = NULL,
  lr = NULL,
  from = NULL,
  to = NULL,
  dmode = NULL,
  pres_qc = NULL,
  temp_qc = NULL,
  psal_qc = NULL,
  doxy_qc = NULL,
  ticket = NULL,
  limit = 10,
  ...
)

argo_files(wmo = NULL, cyc = NULL, ...)

argo_qwmo(qwmo, limit = 10, ...)

argo_plan(...)

argo_buoy_files(dac, id, ...)

argo(dac, id, cycle, dtype, ...)
```

Arguments

func	A function, one of n, np, nf, coord, fullcoord, list, ftplist, ticket, version
of	of string
qwmo	qwmo string
wmo	wmo string. mandatory when using argo_files
box	Bounding box, of the form: A) lon, lat for geographical coordinates of the center of a box, or B) min lon, min lat, width, height, for geographical coordinates of the center of the box and its width and height, and the longitude must given between -180W and 180E. Width and height are in degrees of longitude and latitude.
area	(integer/character), One of 0, 1, or 2, but can be in combination. e.g. 0, '0,2' See Details.

`around` (character) Selects profiles located around a given center point. List of 3 or 4 numerical values depending on how the center point need to be specified: e.g., '-40,35,100', '6900678,2,200', '6900678,2,200,30'. See Details

`year` restrict profiles sampled on a single, or a list of given years. One or a comma separated list of numerical value(s) higher than 0 and lower than 9999.

`yearmin, yearmax`
restrict profiles sampled before (yearmax) and/or after (yearmin) a given year. A numerical value higher than 0 and lower than 9999. cannot be applied with the other restriction parameter `year`

`month` restrict profiles sampled on a single, or a list of given month(s). One or a comma separated list of numerical value(s) higher or equal to 1 and lower or equal to 12. The month convention is a standard one: January is 1, February is 2, ... December is 12.

`monthmin, monthmax`
restrict profiles sampled before (monthmax) and/or after (monthmin) a given month. Higher or equal to 1 and lower or equal to 12. The month convention is a standard one: January is 1, February is 2, ... December is 12. These restrictions cannot be applied with the other restriction parameter month. At this time, these parameters are not circular, so that the restriction chain: monthmin=12&monthmax=2 will through an error and not select December to February profiles. To do so, you need to use a coma separated list of months using the month restriction parameter.

`lr` restriction allows you to impose the last report (hence lr) date in days. A numerical value in days between 1 (profiles sampled yesterday) and 60 (profiles sampled over the last 60 days). This restriction allows a simple selection of the so-called 'active' floats, ie those which reported a profiles over the last 30 days.

`from, to` select profiles sampled before (to) and/or after (from) an explicit date (included). The date is specified following the format: YYYYMMDD, ie the year, month and day numbers.

`dmode` (character) imposes a restriction on the Data Mode of profiles. A single value or a coma separated list of characters defining the Data Mode to select. It can be: R for "Real Time", A for "Real Time with Adjusted value" and D for "Delayed Mode". See Details.

`pres_qc, temp_qc, psal_qc, doxy_qc`
Quality control. Imposes a restriction on the profile data quality flag. For a given variable PARAM which can be: pres (pressure), temp (temperature), psal (salinity) or doxy (oxygen), this restriction selects profiles having one or a coma separated list of data quality flag. See Details.

`ticket` (numeric) select profiles with or without a ticket filled in the database. A value: 0 (no ticket) or 1 (has a ticket). See for more details.

`limit` (integer) number to return

`...` Curl options passed on to `HttpClient`. Optional

`cyc` a cycle number

`dac` (character) Data assembly center code

`id` (numeric) Buoy identifier

`cycle`	(numeric) Cycle number
`dtype`	(character) Data type, one of `D` for delayed, or `R` for real-time

Details

`area` parameter definitions:

- Value 0 selects profiles located in the North-Atlantic ocean north of 20S and not in areas 1 and 2.
- Value 1 selects profiles located in the Mediterranean Sea.
- Value 2 selects profiles located in the Nordic Seas.

`around` parameter definitions:

- Specification 1: The location is specified with specific geographical coordinates in the following format: around=longitude,latitude,distance - The longitude must given between -180W and 180E and the distance is in kilometers.
- Specification 2: The location is the one of an existing profile in the database. It is thus specified with a float WMO and a cycle number: around=wmo,cyc,distance This specification can take an optional fourth value specifying the time range in days around the specified profile.

`dmode` parameter definitions:

- Data from Argo floats are transmitted from the float, passed through processing and automatic quality control procedures. These profiles have a Data Mode called: real-time data.
- The data are also issued to the Principle Investigators who apply other procedures to check data quality returned to the global data centre within 6 to 12 months. These profiles have a Data Mode called: delayed mode data.
- The adjustments applied to delayed-data may also be applied to real-time data, to correct sensor drifts for real-time users. These profiles have a Data Mode called: real time data with adjusted values.

`*_qc` parameter definitions: This information was extracted from the netcdf profile variable PROFILE_PARAM_QC. Once quality control procedures have been applied, a synthetic flag is assigned for each parameter of each profile under this variable in netcdf files. It indicates the fraction n of profile levels with good data. It can take one of the following values:

- A or F: All (n=100%) or none (n=0%) of the profile levels contain good data,
- B,C,D,E: n is in one of the intermediate range: 75-100, 50-75, 25-50 or 0-25
- empty: No QC was performed.

File storage

We use **rappdirs** to store files, see `user_cache_dir` for how we determine the directory on your machine to save files to, and run `rappdirs::user_cache_dir("rnoaa/argo")` to get that directory.

API Status

The API weas down as of 2019-11-07, and probably some time before that. The following functions were defunct:

- argo_search
- argo_files
- argo_qwmo
- argo_plan

These functions are working again as of 2020-06-12.

Examples

```
## Not run:
# Search Argo metadata
## Number of profiles
argo_search("np", limit = 3)
## Number of floats
argo_search("nf", limit = 3)
## Number of both profiles and floats
argo_search("n", limit = 3)
## return the coordinates in time and space of profiles
argo_search("coord", limit = 3)
## return the coordinates in time and space of profiles, plus other metadata
argo_search("fullcoord", limit = 3)

## List various things, e.g,...
### data assembly centers
argo_search("list", "dac")
### data modes
argo_search("list", "dmode", limit = 5)
### World Meteorological Organization unique float ID's
argo_search("list", "wmo", limit = 5)
### Profile years
argo_search("list", "year", limit = 5)

## coord or fullcoord with specific buoy id
argo_search("coord", wmo = 13857, limit = 3)
argo_search("fullcoord", wmo = 13857, limit = 3)

# Spatial search
### search by bounding box (see param def above)
argo_search("coord", box = c(-40, 35, 3, 2))
### search by area
argo_search("coord", area = 0)
### search by around
argo_search("coord", around = '-40,35,100')

# Time based search
### search by year
argo_search("coord", year = 2006)
```

```
### search by yearmin and yearmax
argo_search("coord", yearmin = 2007)
argo_search("coord", yearmin = 2007, yearmax = 2009)
### search by month
argo_search("coord", month = '12,1,2')
### search by from or to
argo_search("coord", from = 20090212)
argo_search("coord", to = 20051129)

# Data mode search
argo_search("coord", dmode = "R")
argo_search("coord", dmode = "R,A")

# Data quality based search
argo_search("coord", pres_qc = "A,B")
argo_search("coord", temp_qc = "A")
argo_search("coord", pres_qc = "A", temp_qc = "A")

# Ticket search
argo_search("coord", ticket = 0)

## Search on partial float id number
argo_qwmo(qwmo = 49)
argo_qwmo(qwmo = 49, limit = 2)

## Get files
argo_files(wmo = 13857)
argo_files(wmo = 13857, cyc = 12)
argo_files(wmo = 13857, cyc = 45)

## Get planned buoys data, accepts no parameters
argo_plan()

# Get files for a buoy, must specify data assembly center (dac)
argo_buoy_files(dac = "bodc", id = 1901309)
argo_buoy_files(dac = "kma", id = 2900308)

# Get data
x <- argo_buoy_files(dac = "meds", id = 4900881)
argo(dac = "meds", id = 4900881, cycle = 127, dtype = "D")

## End(Not run)
```

`autoplot.meteo_coverage` *autoplot method for meteo_coverage objects*

Description

autoplot method for meteo_coverage objects

Usage

```
## S3 method for class 'meteo_coverage'
autoplot(mateo_coverage, old_style = FALSE)
```

Arguments

mateo_coverage	the object returned from meteo_coverage()
old_style	(logical) create the old style of plots, which is faster, but does not plot gaps to indicate missing data

Details

see meteo_coverage() for examples

Value

A ggplot2 plot

---

bsw *Blended sea winds (BSW)*

Description

The Blended Sea Winds dataset contains globally gridded, high-resolution ocean surface vector winds and wind stresses on a global 0.25° grid, and multiple time resolutions of six-hourly, daily, monthly, and 11-year (1995–2005) climatological monthlies.

Usage

```
bsw(date = NULL, uv_stress = "uv", resolution = "6hrly", ...)
```

Arguments

date	(date/character) date, in the form YYYY-MM-DD if resolution is 6hrly or daily, or in the form YYYY-MM if resolution is monthly. For resolution=clm can be left NULL. If given, must be in the range 1987-07-09 to today-1 (yesterday)
uv_stress	(character) one of uv or stresss, not sure what these mean exactly yet. Default: uv
resolution	(character) temporal resolution. one of 6hrly, clm, daily, or monthly. See Details.
...	curl options passed on to crul::verb-GET

Details

Products are available from July 9th, 1987 - present.

Uses ncdf4 under the hood to read NetCDF files

Value

an object of class ncdf4

Citing NOAA and BSW data

Message from NOAA: "We also ask you to acknowledge us in your use of the data to help us justify continued service. This may be done by including text such as: The wind data are acquired from NOAA's National Climatic Data Center, via their website We would also appreciate receiving a copy of the relevant publication."

Temporal resolution

- 6hrly: 6-hourly, 4 global snapshots (u,v) at UTC 00, 06, 12 and 18Z
- clm: climatological monthlies; also provided is the scalar mean (u,v,w)
- daily: averages of the 6hrly time points, thus with a center time 09Z; also provided is the scalar mean, (u,v,w)
- monthly: averages of daily data; also provided is the scalar mean (u,v,w)

Note

See bsw_cache for managing cached files

We only handle the netcdf files for now, we're avoiding the ieee files, see

Examples

```
## Not run:
# 6hrly data
## uv
x <- bsw(date = "2017-10-01")
## stress
y <- bsw(date = "2011-08-01", uv_stress = "stress")

# daily
z <- bsw(date = "2017-10-01", resolution = "daily")

# monthly
w <- bsw(date = "2011-08", resolution = "monthly")

# clm
# x <- bsw(resolution = "clm")

## End(Not run)
```

buoy *Get NOAA buoy data from the National Buoy Data Center*

Description

Get NOAA buoy data from the National Buoy Data Center

Usage

```
buoy(dataset, buoyid, year = NULL, datatype = NULL, ...)

buoys(dataset)

buoy_stations(refresh = FALSE, ...)
```

Arguments

`dataset`	(character) Dataset name to query. See below for Details. Required
`buoyid`	Buoy ID, can be numeric/integer/character. Required
`year`	(integer) Year of data collection. Optional. Note there is a special value `9999` that, if found, contains the most up to date data.
`datatype`	(character) Data type, one of 'c', 'cc', 'p', 'o'. Optional
`...`	Curl options passed on to crul::verb-GET Optional. A number of different HTTP requests are made internally, but we only pass this on to the request to get the netcdf file in the internal function `get_ncdf_file()`
`refresh`	(logical) Whether to use cached data (`FALSE`) or get new data (`FALSE`). Default: `FALSE`

Details

Functions:

- buoy_stations - Get buoy stations. A cached version of the dataset is available in the package. Beware, takes a long time to run if you do `refresh = TRUE`
- buoys - Get available buoys given a dataset name
- buoy - Get data given some combination of dataset name, buoy ID, year, and datatype

Options for the dataset parameter. One of:

- adcp - Acoustic Doppler Current Profiler data
- adcp2 - MMS Acoustic Doppler Current Profiler data
- cwind - Continuous Winds data
- dart - Deep-ocean Assessment and Reporting of Tsunamis data
- mmbcur - Marsh-McBirney Current Measurements data

- ocean - Oceanographic data
- pwind - Peak Winds data
- stdmet - Standard Meteorological data
- swden - Spectral Wave Density data with Spectral Wave Direction data
- wlevel - Water Level data

Value

If netcdf data has lat/lon variables, then we'll parse into a tidy data.frame. If not, we'll give back the ncdf4 object for the user to parse (in which case the data.frame will be empty).

Examples

```
## Not run:
if (crul::ok("https://dods.ndbc.noaa.gov/thredds", timeout_ms = 1000)) {

# Get buoy station information
x <- buoy_stations()
# refresh stations as needed, takes a while to run
# you shouldn't need to update very often
# x <- buoy_stations(refresh = TRUE)
if (interactive() && requireNamespace("leaflet")){
library("leaflet")
z <- leaflet(data = na.omit(x))
z <- leaflet::addTiles(z)
leaflet::addCircles(z, ~lon, ~lat, opacity = 0.5)
}

# year=9999 to get most current data - not always available
buoy(dataset = "swden", buoyid = 46012, year = 9999)

# Get available buoys
buoys(dataset = 'cwind')

# Get data for a buoy
## if no year or datatype specified, we get the first file
buoy(dataset = 'cwind', buoyid = 46085)

# Including specific year
buoy(dataset = 'cwind', buoyid = 41001, year = 1999)

# Including specific year and datatype
buoy(dataset = 'cwind', buoyid = 45005, year = 2008, datatype = "c")
buoy(dataset = 'cwind', buoyid = 41001, year = 1997, datatype = "c")

# Other datasets
buoy(dataset = 'ocean', buoyid = 41029)

# curl debugging
buoy(dataset = 'cwind', buoyid = 46085, verbose = TRUE)
```

```
# some buoy ids are character, case doesn't matter, we'll account for it
buoy(dataset = "stdmet", buoyid = "VCAF1")
buoy(dataset = "stdmet", buoyid = "wplf1")
buoy(dataset = "dart", buoyid = "dartu")

}

## End(Not run)
```

---

coops *Get NOAA co-ops data*

Description

Get NOAA co-ops data

Usage

```
coops_search(
  begin_date = NULL,
  end_date = NULL,
  station_name = NULL,
  product,
  datum = NULL,
  units = "metric",
  time_zone = "gmt",
  application = "rnoaa",
  ...
)
```

Arguments

begin_date	(numeric) Date in yyyymmdd format. Required
end_date	(numeric) Date in yyyymmdd format. Required
station_name	(numeric) a station name. Required
product	(character) Specify the data type. See below for Details. Required
datum	(character) See below for Details. Required for all water level products.
units	(character) Specify metric or english (imperial) units, one of 'metric', 'english'.
time_zone	(character) Time zone, one of 'gmt', 'lst', 'lst_ldt'. For GMT, we convert time stamps to GMT. For LST, we look up the time zone of the station with its lat/lon values, and assign that time zone. When `product="predictions"` we don't adjust times at all.
application	(character) If called within an external package, set to the name of your organization. Optional
...	Curl options passed on to crul::verb-GET Optional

Details

Options for the product paramter. One of:

- water_level - Preliminary or verified water levels, depending on availability
- air_temperature - Air temperature as measured at the station
- water_temperature - Water temperature as measured at the station
- wind - Wind speed, direction, and gusts as measured at the station
- air_pressure - Barometric pressure as measured at the station
- air_gap - Air Gap (distance between a bridge and the water's surface) at the station
- conductivity - The water's conductivity as measured at the station
- visibility - Visibility from the station's visibility sensor. A measure of atmospheric clarity
- humidity - Relative humidity as measured at the station
- salinity - Salinity and specific gravity data for the station
- one_minute_water_level - One minute water level data for the station
- predictions - 6 minute predictions water level data for the station
- hourly_height - Verified hourly height water level data for the station
- high_low - Verified high/low water level data for the station
- daily_mean - Verified daily mean water level data for the station
- monthly_mean - Verified monthly mean water level data for the station
- datums - datums data for the stations
- currents - Currents data for currents stations

Maximum Durations in a Single Call:

- Products water_level through predictions allow requests for up to
- Products hourly_height and high_low allow requests for up to
- Products daily_mean and monthly_mean allow requests for up to

Options for the datum parameter. One of:

- MHHW - Mean higher high water
- MHW - Mean high water
- MTL - Mean tide level
- MSL - Mean sea level
- MLW - Mean low water
- MLLW - Mean lower low water
- NAVD - North American Vertical Datum
- STND - Station datum

Value

List, of length one or two.

- metadata A list of metadata with slots id, name, lat, lon
- data A data.frame with data

Author(s)

Scott Chamberlain, Joseph Stachelek, Tom Philippi

Examples

```
## Not run:
# Get monthly mean sea level data at Vaca Key (8723970)
coops_search(station_name = 8723970, begin_date = 20120301,
  end_date = 20141001, datum = "stnd", product = "monthly_mean")

# Get verified water level data at Vaca Key (8723970)
coops_search(station_name = 8723970, begin_date = 20140927,
  end_date = 20140928, datum = "stnd", product = "water_level")

# Get daily mean water level data at Fairport, OH (9063053)
coops_search(station_name = 9063053, begin_date = 20150927,
  end_date = 20150928, product = "daily_mean", datum = "stnd",
  time_zone = "lst")

# Get air temperature at Vaca Key (8723970)
coops_search(station_name = 8723970, begin_date = 20140927,
  end_date = 20140928, product = "air_temperature")

# Get water temperature at Vaca Key (8723970)
coops_search(station_name = 8723970, begin_date = 20140927,
  end_date = 20140928, product = "water_temperature")

# Get air pressure at Vaca Key (8723970)
coops_search(station_name = 8723970, begin_date = 20140927,
  end_date = 20140928, product = "air_pressure")

# Get wind at Vaca Key (8723970)
coops_search(station_name = 8723970, begin_date = 20140927,
  end_date = 20140928, product = "wind")

# Get hourly water level height at Key West (8724580)
coops_search(station_name = 8724580, begin_date = 20140927,
  end_date = 20140928, product = "hourly_height", datum = "stnd")

# Get high-low water level at Key West (8724580)
coops_search(station_name = 8724580, begin_date = 20140927,
  end_date = 20140928, product = "high_low", datum = "stnd")

# Get currents data at Pascagoula Harbor (ps0401)
coops_search(station_name = "ps0401", begin_date = 20151221,
```

```
  end_date = 20151222, product = "currents")

# Get one-minute water level at Vaca Key (8723970)
coops_search(station_name = 8723970, begin_date = 20140927,
  end_date = 20140928, datum = "stnd", product = "one_minute_water_level")

# Get datums at Fort Myers, FL (8725520)
coops_search(station_name = 8725520, product = "datums")

# Get water level predictions at Vaca Key (8723970)
coops_search(station_name = 8723970, begin_date = 20140928,
  end_date = 20140929, datum = "stnd", product = "predictions")


## End(Not run)
```

`cpc_prcp` *Precipitation data from NOAA Climate Prediction Center (CPC)*

Description

Precipitation data from NOAA Climate Prediction Center (CPC)

Usage

```
cpc_prcp(date, us = FALSE, drop_undefined = FALSE, ...)
```

Arguments

`date`	(date/character) date in YYYY-MM-DD format
`us`	(logical) US data only? default: `FALSE`
`drop_undefined`	(logical) drop undefined precipitation values (values in the `precip` column in the output data.frame). default: `FALSE`
`...`	curl options passed on to crul::verb-GET

Details

Rainfall data for the world (1979-present, resolution 50 km), and the US (1948-present, resolution 25 km).

Value

a data.frame, with columns:

- lon - longitude (0 to 360)
- lat - latitude (-90 to 90)
- precip - precipitation (in mm) (see Details for more information)

Data processing in this function

Internally we multiply all precipitation measurements by 0.1 as per the CPC documentation.

Values of -99.0 are classified as "undefined". These values can be removed by setting `drop_undefined = TRUE` in the `cpc_prcp` function call. These undefined values are not dropped by default - so do remember to set `drop_undefined = TRUE` to drop them; or you can easily do it yourself by e.g., `subset(x,precip >= 0)`

Note

See cpc_cache for managing cached files

Examples

```
## Not run:
x = cpc_prcp(date = "2017-01-15")
cpc_prcp(date = "2015-06-05")
cpc_prcp(date = "2017-01-15")
cpc_prcp(date = "2005-07-09")
cpc_prcp(date = "1979-07-19")

# United States data only
cpc_prcp(date = "2005-07-09", us = TRUE)
cpc_prcp(date = "2009-08-03", us = TRUE)
cpc_prcp(date = "1998-04-23", us = TRUE)

# drop undefined values (those given as -99.0)
cpc_prcp(date = "1998-04-23", drop_undefined = TRUE)

## End(Not run)
```

---

ersst *NOAA Extended Reconstructed Sea Surface Temperature (ERSST) data*

Description

NOAA Extended Reconstructed Sea Surface Temperature (ERSST) data

Usage

```
ersst(year, month, overwrite = TRUE, version = "v5", ...)
```

Arguments

`year`	(numeric) A year. Must be > 1853. The max value is whatever the current year is. Required
`month`	A month, character or numeric. If single digit (e.g. 8), we add a zero in front (e.g., 08). Required

`overwrite`	(logical) To overwrite the path to store files in or not, Default: `TRUE`
`version`	(character) ERSST version. one of "v5" (default) or "v4"
`...`	Curl options passed on to crul::verb-GET

Details

See ersst_cache for managing cached files

`ersst()` currently defaults to use ERSST v5 - you can set v4 or v5 using the `version` parameter

If a request is unsuccesful, the file written to disk is deleted before the function exits.

If you use this data in your research please cite rnoaa (`citation("rnoaa")`), and cite NOAA ERSST (see citations at link above)

Value

An `ncdf4` object. See **ncdf4** for parsing the output

Examples

```
## Not run:
# October, 2015
ersst(year = 2015, month = 10)

# May, 2015
ersst(year = 2015, month = 5)
ersst(year = 2015, month = "05")

# February, 1890
ersst(year = 1890, month = 2)

# Process data
library("ncdf4")
res <- ersst(year = 1890, month = 2)
## varibles
names(res$var)
## get a variable
ncdf4::ncvar_get(res, "ssta")

## End(Not run)
```

`fipscodes` *FIPS codes for US states.*

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.

---

ghcnd *Get all GHCND data from a single weather site*

Description

This function uses ftp to access the Global Historical Climatology Network daily weather data from NOAA's FTP server for a single weather site. It requires the site identification number for that site and will pull the entire weather dataset for the site.

Usage

```
ghcnd(stationid, refresh = FALSE, ...)

ghcnd_read(path, ...)
```

Arguments

stationid	(character) A character vector giving the identification of the weather stations for which the user would like to pull data. To get a full and current list of stations, the user can use the ghcnd_stations() function. To identify stations within a certain radius of a location, the user can use the meteo_nearby_stations() function.
refresh	(logical) If TRUE force re-download of data. Default: FALSE
...	In the case of ghcnd() additional curl options to pass through to crul::HttpClient. In the case of ghcnd_read further options passed on to read.csv
path	(character) a path to a file with a .dly extension - already downloaded on your computer

Details

This function saves the full set of weather data for the queried site locally in the directory specified by the path argument.

You can access the path for the cached file via `attr(x,"source")`

You can access the last modified time for the cached file via `attr(x,"file_modified")`

Messages are printed to the console about file path and file last modified time which you can suppress with `suppressMessages()`

For those station ids that are not found, we will delete the file locally so that a bad station id file is not cached. The returned data for a bad station id will be an empty data.frame and the attributes are empty strings.

Value

A tibble (data.frame) which contains data pulled from NOAA's FTP server for the queried weather site. A README file with more information about the format of this file is available from NOAA . This file is formatted so each line of the file gives the daily weather observations for a single weather variable for all days of one month of one year. In addition to measurements, columns are included for certain flags, which add information on observation sources and quality and are further explained in NOAA's README file for the data.

Base URL

The base url for data requests can be changed. The allowed urls are: (default)

You can set the base url using the `RNOAA_GHCND_BASE_URL` environment variable; see example below.

The reason for this is that sometimes one base url source is temporarily down, but another base url may work. It doesn't make sense to allow an arbitrary base URL; open an issue if there is another valid base URL for GHNCD data that we should add to the allowed set of base urls.

Note

See ghcnd_cache for managing cached files

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>, Adam Erickson <adam.erickson@ubc.ca>

See Also

To generate a weather dataset for a single weather site that has been cleaned to a tidier weather format, the user should use the `ghcnd_search()` function, which calls `ghcnd()` and then processes the output, or `meteo_tidy_ghcnd()`, which wraps the `ghcnd_search()` function to output a tidy dataframe. To pull GHCND data from multiple monitors, see `meteo_pull_monitors()`

Examples

```
## Not run:
# Get data
ghcnd(stationid = "AGE00147704")

stations <- ghcnd_stations()
ghcnd(stations$id[40])

library("dplyr")
ghcnd(stations$id[80300]) %>% select(id, element) %>% slice(1:3)

# manipulate data
## using built in fxns
dat <- ghcnd(stationid = "AGE00147704")
(alldat <- ghcnd_splitvars(dat))

## using dplyr
library("dplyr")
dat <- ghcnd(stationid = "AGE00147704")
filter(dat, element == "PRCP", year == 1909)

# refresh the cached file
ghcnd(stationid = "AGE00147704", refresh = TRUE)

# Read in a .dly file you've already downloaded
path <- system.file("examples/AGE00147704.dly", package = "rnoaa")
ghcnd_read(path)

# change the base url for data requests
Sys.setenv(RNOAA_GHCND_BASE_URL =
  "ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/all")
ghcnd(stations$id[45], verbose = TRUE)
## must be in the allowed set of urls
# Sys.setenv(RNOAA_GHCND_BASE_URL = "https://google.com")
# ghcnd(stations$id[58], verbose = TRUE)

## End(Not run)
```

---

ghcnd_search *Get a cleaned version of GHCND data from a single weather site*

Description

This function uses ftp to access the Global Historical Climatology Network daily weather data from NOAA's FTP server for a single weather monitor site. It requires the site identification number for that site and will pull the entire weather dataset for the site. It will then clean this data to convert it to a tidier format and will also, if requested, filter it to a certain date range and to certain weather variables.

Usage

```
ghcnd_search(
  stationid,
  date_min = NULL,
  date_max = NULL,
  var = "all",
  refresh = FALSE,
  ...
)
```

Arguments

stationid	(character) A character vector giving the identification of the weather stations for which the user would like to pull data. To get a full and current list of stations, the user can use the ghcnd_stations() function. To identify stations within a certain radius of a location, the user can use the meteo_nearby_stations() function.
date_min	A character string giving the earliest date of the daily weather time series that the user would like in the final output. This character string should be formatted as "yyyy-mm-dd". If not specified, the default is to keep all daily data for the queried weather site from the earliest available date.
date_max	A character string giving the latest date of the daily weather time series that the user would like in the final output. This character string should be formatted as "yyyy-mm-dd". If not specified, the default is to keep all daily data for the queried weather site through the most current available date.
var	A character vector specifying either "all" (pull all available weather parameters for the site) or the weather parameters to keep in the final data (e.g., c("TMAX","TMIN") to only keep maximum and minimum temperature). Example choices for this argument include: • PRCP: Precipitation, in tenths of millimeters • TAVG: Average temperature, in tenths of degrees Celsius • TMAX: Maximum temperature, in tenths of degrees Celsius • TMIN: Minimum temperature, in tenths of degrees Celsius A full list of possible weather variables is available in NOAA's README file for the GHCND data . Most weather stations will only have a small subset of all the possible weather variables, so the data generated by this function may not include all of the variables the user specifies through this argument.
refresh	(logical) If TRUE force re-download of data. Default: FALSE
...	In the case of ghcnd() additional curl options to pass through to crul::HttpClient. In the case of ghcnd_read further options passed on to read.csv

Details

Messages are printed to the console about file path, file last modified time which you can suppress with suppressMessages()

Value

A list object with slots for each of the available specified weather variables. Each element in the list is a separate time series dataframe with daily observations, as well as flag values, for one of the weather variables. The flag values give information on the quality and source of each observation; see the NOAA README file linked above for more information. Each data.frame is sorted by date, with the earliest date first.

Note

This function calls ghcnd(), which will download and save data from all available dates and weather variables for the queried weather station. The step of limiting the dataset to only certain dates and / or weather variables, using the date_min, date_max, and var arguments, does not occur until after the full data has been pulled.

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>, Adam Erickson <adam.erickson@ubc.ca>

See Also

meteo_pull_monitors(), meteo_tidy_ghcnd()

Examples

```
## Not run:
# Search based on variable and/or date
ghcnd_search("AGE00147704", var = "PRCP")
ghcnd_search("AGE00147704", var = "PRCP", date_min = "1920-01-01")
ghcnd_search("AGE00147704", var = "PRCP", date_max = "1915-01-01")
ghcnd_search("AGE00147704", var = "PRCP", date_min = "1920-01-01",
             date_max = "1925-01-01")
ghcnd_search("AGE00147704", date_min = "1920-01-01", date_max = "1925-01-01")
ghcnd_search("AGE00147704", var = c("PRCP","TMIN"))
ghcnd_search("AGE00147704", var = c("PRCP","TMIN"), date_min = "1920-01-01")
ghcnd_search("AGE00147704", var = "adfdf")

# refresh the cached file
ghcnd_search("AGE00147704", var = "PRCP", refresh = TRUE)

## End(Not run)
```

---

ghcnd_splitvars *Split variables in data returned from* ghcnd

---

Description

This function is a helper function for ghcnd_search(). It helps with cleaning up the data returned from ghcnd(), to get it in a format that is easier to work with.

Usage

```
ghcnd_splitvars(x)
```

Arguments

x	An object returned from ghcnd()

Note

See ghcnd() examples

Author(s)

Scott Chamberlain, Adam Erickson, Elio Campitelli

---

ghcnd_states *Get meta-data on the GHCND daily data*

Description

These function allow you to pull the current versions of certain meta-datasets for the GHCND, including lists of country and state abbreviations used in some of the weather station IDs and information about the current version of the data.

Usage

```
ghcnd_states(...)

ghcnd_countries(...)

ghcnd_version(...)
```

Arguments

...	In the case of ghcnd() additional curl options to pass through to crul::HttpClient. In the case of ghcnd_read further options passed on to read.csv

Details

Functions:

- ghcnd_version: Get current version of GHCND data
- ghcnd_states: Get US/Canada state names and 2-letter codes
- ghcnd_countries: Get country names and 2-letter codes

Author(s)

Scott Chamberlain <myrmecocystus@gmail.com>, Adam Erickson <adam.erickson@ubc.ca>

Examples

```
## Not run:
ghcnd_states()
ghcnd_countries()
ghcnd_version()

## End(Not run)
```

`ghcnd_stations` *Get information on the GHCND weather stations*

Description

This function returns an object with a dataframe with meta-information about all available GHCND weather stations.

Usage

```
ghcnd_stations(refresh = FALSE, ...)
```

Arguments

`refresh`	(logical) If `TRUE` force re-download of data. Default: `FALSE`
`...`	In the case of `ghcnd()` additional curl options to pass through to crul::HttpClient. In the case of `ghcnd_read` further options passed on to `read.csv`

Value

This function returns a tibble (dataframe) with a weather station on each row with the following columns:

- `id`: The weather station's ID number. The first two letters denote the country (using FIPS country codes).
- `latitude`: The station's latitude, in decimal degrees. Southern latitudes will be negative.
- `longitude`: The station's longitude, in decimal degrees. Western longitudes will be negative.
- `elevation`: The station's elevation, in meters.
- `name`: The station's name.
- `gsn_flag`: "GSN" if the monitor belongs to the GCOS Surface Network (GSN). Otherwise either blank or missing.
- `wmo_id`: If the station has a WMO number, this column gives that number. Otherwise either blank or missing.
- `element`: A weather variable recorded at some point during that station's history. See the link below in "References" for definitions of the abbreviations used for this variable.
- `first_year`: The first year of data available at that station for that weather element.
- `last_year`: The last year of data available at that station for that weather element.

If a weather station has data on more than one weather variable, it will be represented in multiple rows of this output dataframe.

Note

Since this function is pulling a large dataset by ftp, it may take a while to run.

References

For more documentation on the returned dataset, see

Examples

```
## Not run:
# Get stations, ghcnd-stations and ghcnd-inventory merged
(stations <- ghcnd_stations())

library(dplyr)
# filter by state
stations %>% filter(state == "IL")
stations %>% filter(state == "OR")
# those without state values
stations %>% filter(state == "")
# filter by element
stations %>% filter(element == "PRCP")
# filter by id prefix
stations %>% filter(grepl("^AF", id))
stations %>% filter(grepl("^AFM", id))
# filter by station long name
stations %>% filter(name == "CALLATHARRA")

## End(Not run)
```

---

homr *Historical Observing Metadata Repository (HOMR) station metadata*

Description

Historical Observing Metadata Repository (HOMR) station metadata

Usage

```
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

qid	One of COOP, FAA, GHCND, ICAO, NCDCSTNID, NWSLI, TRANS, WBAN, or WMO, or any of those plus a-z0-9, or just a-z0-9. (qid = qualified ID)
qidMod	(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.
station	(character) A station id.
state	(character) A two-letter state abbreviation. Two-letter code for US states, Canadian provinces, and other Island areas.
county	(character) A two letter county code. US county names, best used with a state identifier.
country	(character) A two letter country code. See here for a list of valid country names.
name	(character) One of 0-9A-Z+. Searches on any type of name we have for the station.
nameMod	(character) is|starts|ends|contains. Specifies how the name parameter should be applied within the search. If a name is passed but the nameMod parameter is not used, nameMod is assumed to be IS.
platform	(character) (aka network) ASOS|USCRN|USHCN|NEXRAD|AL USRCRN|USRCRN|COOP. Limit the search to stations of a certain platform/network type.
date	(character) YYYY-MM-DD|all Limits values to only those that occurred on a specific date. Alternatively, date=all will return all values for matched stations. If this field is omitted, the search will return only the most recent values for each field.
begindate, enddate	YYYY-MM-DD. Limits values to only those that occurred within a date range.
headersOnly	(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, 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.
...	Curl options passed on to crul::verb-GET (optional)

Details

Since the definitions for variables are always the same, we don't include the ability to get description data in this function. Use `homr_definitions()` to get descriptions information.

Value

A list, with elements named by the station ids.

Examples

```
## Not run:
homr(qid = 'COOP:046742')
homr(qid = ':046742')
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', begindate='2005-01-01', enddate='2011-01-01')
homr(state='DE', headersOnly=TRUE)
homr(station=20002078)
homr(station=20002078, date='all', phrData=FALSE)

# Optionally pass in curl options
homr(headersOnly=TRUE, state='NC', county='BUNCOMBE', verbose = TRUE)

## End(Not run)
```

---

homr_definitions *Historical Observing Metadata Repository (HOMR) station metadata - definitions*

Description

Historical Observing Metadata Repository (HOMR) station metadata - definitions

Usage

```
homr_definitions(...)
```

Arguments

...	Curl options passed on to crul::verb-GET optional

Examples

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

## End(Not run)
```

`isd` *Get and parse NOAA ISD/ISH data*

Description

Get and parse NOAA ISD/ISH data

Usage

```
isd(
  usaf,
  wban,
  year,
  overwrite = TRUE,
  cleanup = TRUE,
  additional = TRUE,
  parallel = FALSE,
  cores = getOption("cl.cores", 2),
  progress = FALSE,
  force = FALSE,
  ...
)
```

Arguments

usaf, wban	(character) USAF and WBAN code. Required
year	(numeric) One of the years from 1901 to the current year. Required.
overwrite	(logical) To overwrite the path to store files in or not, Default: TRUE
cleanup	(logical) If TRUE, remove compressed .gz file at end of function execution. Processing data takes up a lot of time, so we cache a cleaned version of the data. Cleaning up will save you on disk space. Default: TRUE
additional	(logical) include additional and remarks data sections in output. Default: TRUE. Passed on to isdparser::isd_parse()
parallel	(logical) do processing in parallel. Default: FALSE
cores	(integer) number of cores to use: Default: 2. We look in your option "cl.cores", but use default value if not found.

progress	(logical) print progress - ignored if parallel=TRUE. The default is FALSE because printing progress adds a small bit of time, so if processing time is important, then keep as FALSE
force	(logical) force download? Default: FALSE We use a cached version (an .rds compressed file) if it exists, but this will override that behavior.
...	Curl options passed on to crul::verb-GET

Details

isd saves the full set of weather data for the queried site locally in the directory specified by the path argument. You can access the path for the cached file via attr(x,"source")

We use **isdparser** internally to parse ISD files. They are relatively complex to parse, so a separate package takes care of that.

This function first looks for whether the data for your specific query has already been downloaded previously in the directory given by the path parameter. If not found, the data is requested form NOAA's FTP server. The first time a dataset is pulled down we must a) download the data, b) process the data, and c) save a compressed .rds file to disk. The next time the same data is requested, we only have to read back in the .rds file, and is quite fast. The benfit of writing to .rds files is that data is compressed, taking up less space on your disk, and data is read back in quickly, without changing any data classes in your data, whereas we'd have to jump through hoops to do that with reading in csv. The processing can take quite a long time since the data is quite messy and takes a bunch of regex to split apart text strings. We hope to speed this process up in the future. See examples below for different behavior.

Value

A tibble (data.frame).

Errors

Note that when you get an error similar to Error: download failed for ftp://ftp.ncdc.noaa.gov/pub/data/noaa/1955/011490-99999-1955.gz, the file does not exist on NOAA's ftp servers. If your internet is down, you'll get a different error.

Note

There are now no transformations (scaling, class changes, etc.) done on the output data. This may change in the future with parameters to toggle transformations, but none are done for now. See isdparser::isd_transform() for transformation help. Comprehensive transformations for all variables are not yet available but should be available in the next version of this package.

See isd_cache for managing cached files

References

ftp://ftp.ncdc.noaa.gov/pub/data/noaa/

See Also

Other isd: isd_read(), isd_stations_search(), isd_stations()

Examples

```
## Not run:
# Get station table
(stations <- isd_stations())

## plot stations
### remove incomplete cases, those at 0,0
df <- stations[complete.cases(stations$lat, stations$lon), ]
df <- df[df$lat != 0, ]
### make plot
library("leaflet")
leaflet(data = df) %>%
  addTiles() %>%
  addCircles()

# Get data
(res <- isd(usaf='011490', wban='99999', year=1986))
(res <- isd(usaf='011690', wban='99999', year=1993))
(res <- isd(usaf='109711', wban=99999, year=1970))

# "additional" and "remarks" data sections included by default
# can toggle that parameter to not include those in output, saves time
(res1 <- isd(usaf='011490', wban='99999', year=1986, force = TRUE))
(res2 <- isd(usaf='011490', wban='99999', year=1986, force = TRUE,
  additional = FALSE))

# The first time a dataset is requested takes longer
system.time( isd(usaf='782680', wban='99999', year=2011) )
system.time( isd(usaf='782680', wban='99999', year=2011) )

# Plot data
## get data for multiple stations
res1 <- isd(usaf='011690', wban='99999', year=1993)
res2 <- isd(usaf='782680', wban='99999', year=2011)
res3 <- isd(usaf='008415', wban='99999', year=2016)
res4 <- isd(usaf='109711', wban=99999, year=1970)
## combine data
library(dplyr)
res_all <- bind_rows(res1, res2, res3, res4)
# add date time
library("lubridate")
dd <- sprintf('%s %s', as.character(res_all$date), res_all$time)
res_all$date_time <- ymd_hm(dd)
## remove 999's
res_all <- filter(res_all, temperature < 900)

## plot
if (interactive()) {
  library(ggplot2)
  ggplot(res_all, aes(date_time, temperature)) +
    geom_line() +
    facet_wrap(~usaf_station, scales = 'free_x')
```

```
}

# print progress
## note: if the file is already on your system, you'll see no progress bar
(res <- isd(usaf='011690', wban='99999', year=1993, progress=TRUE))

# parallelize processing
# (res <- isd(usaf=172007, wban=99999, year=2016, parallel=TRUE))

## End(Not run)
```

`isd_read` *Read NOAA ISD/ISH local file*

Description

Read NOAA ISD/ISH local file

Usage

```
isd_read(
  path,
  additional = TRUE,
  parallel = FALSE,
  cores = getOption("cl.cores", 2),
  progress = FALSE
)
```

Arguments

`path`	(character) path to the file. required.
`additional`	(logical) include additional and remarks data sections in output. Default: `TRUE`. Passed on to `isdparser::isd_parse()`
`parallel`	(logical) do processing in parallel. Default: `FALSE`
`cores`	(integer) number of cores to use: Default: 2. We look in your option "cl.cores", but use default value if not found.
`progress`	(logical) print progress - ignored if `parallel=TRUE`. The default is `FALSE` because printing progress adds a small bit of time, so if processing time is important, then keep as `FALSE`

Details

`isd_read` - read a `.gz` file as downloaded from NOAA's website

Value

A tibble (data.frame)

References

ftp://ftp.ncdc.noaa.gov/pub/data/noaa/

See Also

`isd()`, `isd_stations()`, `isd_stations_search()`

Other isd: `isd_stations_search()`, `isd_stations()`, `isd()`

Examples

```
## Not run:
file <- system.file("examples", "011490-99999-1986.gz", package = "rnoaa")
isd_read(file)
isd_read(file, additional = FALSE)

## End(Not run)
```

`isd_stations` *Get NOAA ISD/ISH station data from NOAA FTP server.*

Description

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

Usage

```
isd_stations(refresh = FALSE)
```

Arguments

`refresh`	(logical) Download station data from NOAA ftp server again. Default: `FALSE`

Details

The data table is cached, but you can force download of data from NOAA by setting `refresh=TRUE`

Value

a tibble (data.frame) with the columns:

- usaf - USAF number, character
- wban - WBAN number, character
- station_name - station name, character
- ctry - Country, if given, character
- state - State, if given, character
- icao - ICAO number, if given, character

- lat - Latitude, if given, numeric
- lon - Longitude, if given, numeric
- elev_m - Elevation, if given, numeric
- begin - Begin date of data coverage, of form YYYYMMDD, numeric
- end - End date of data coverage, of form YYYYMMDD, numeric

Note

See isd_cache for managing cached files

References

ftp://ftp.ncdc.noaa.gov/pub/data/noaa/

See Also

Other isd: `isd_read()`, `isd_stations_search()`, `isd()`

Examples

```
## Not run:
# Get station table
(stations <- isd_stations())

## plot stations
### remove incomplete cases, those at 0,0
df <- stations[complete.cases(stations$lat, stations$lon), ]
df <- df[df$lat != 0, ]
### make plot
library("leaflet")
leaflet(data = df) %>%
  addTiles() %>%
  addCircles()

## End(Not run)
```

---

isd_stations_search *Search for NOAA ISD/ISH station data from NOAA FTP server.*

Description

Search for NOAA ISD/ISH station data from NOAA FTP server.

Usage

```
isd_stations_search(lat = NULL, lon = NULL, radius = NULL, bbox = NULL)
```

Arguments

lat	(numeric) Latitude, in decimal degree
lon	(numeric) Latitude, in decimal degree
radius	(numeric) Radius (in km) to search from the lat,lon coordinates
bbox	(numeric) Bounding box, of the form: min-longitude, min-latitude, max-longitude, max-latitude

Details

We internally call `isd_stations()` to get the data.frame of ISD stations, which is quite fast as long as it's not the first time called since we cache the table. Before searching, we clean up the data.frame, removing stations with no lat/long coordinates, those with impossible lat/long coordinates, and those at 0,0.

When lat/lon/radius input we use `meteo_distance()` to search for stations, while when bbox is input, we simply use `dplyr::filter()`

Value

a data.frame with the columns:

- usaf - USAF number, character
- wban - WBAN number, character
- station_name - station name, character
- ctry - Country, if given, character
- state - State, if given, character
- icao - ICAO number, if given, character
- lat - Latitude, if given, numeric
- lon - Longitude, if given, numeric
- elev_m - Elevation, if given, numeric
- begin - Begin date of data coverage, of form YYYYMMDD, numeric
- end - End date of data coverage, of form YYYYMMDD, numeric
- distance - distance (km) (only present if using lat/lon/radius parameter combination)

References

ftp://ftp.ncdc.noaa.gov/pub/data/noaa/

See Also

Other isd: `isd_read()`, `isd_stations()`, `isd()`

Examples

```
## Not run:
## lat, long, radius
isd_stations_search(lat = 38.4, lon = -123, radius = 250)

x <- isd_stations_search(lat = 60, lon = 18, radius = 200)

if (requireNamespace("leaflet")) {
  library("leaflet")
  leaflet() %>%
    addTiles() %>%
    addCircles(lng = x$lon,
               lat = x$lat,
               popup = x$station_name) %>%
    clearBounds()
}

## bounding box
bbox <- c(-125.0, 38.4, -121.8, 40.9)
isd_stations_search(bbox = bbox)

## End(Not run)
```

---

lcd *Local Climitalogical Data from NOAA*

Description

Local Climitalogical Data from NOAA

Usage

```
lcd(station, year, ...)
```

Arguments

station	(character) station code, e.g., "02413099999". we will allow integer/numeric passed here, but station ids can have leading zeros, so it's a good idea to keep stations as character class. required
year	(integer) year, e.g., 2017. required
...	curl options passed on to crul::verb-GET

Value

a data.frame with many columns. the first 10 are metadata:

- station
- date

- latitude
- longitude
- elevation
- name
- report_type
- source

And the rest should be all data columns. The first part of many column names is the time period, being one of:

- hourly
- daily
- monthly
- shortduration

So the variable you are looking for may not be the first part of the column name

Note

See lcd_cache for managing cached files

References

Docs:

Data comes from:

Examples

```
## Not run:
x = lcd(station = "01338099999", year = 2017)
lcd(station = "01338099999", year = 2015)
lcd(station = "02413099999", year = 2009)
lcd(station = "02413099999", year = 2001)

# pass curl options
lcd(station = "02413099999", year = 2002, verbose = TRUE)

## End(Not run)
```

meteo_clear_cache *Clear* meteo *cached files*

Description

The *meteo* functions use an aplication

Usage

```
meteo_clear_cache(force = FALSE)
```

Arguments

force	(logical) force delete. default: FALSE

Note

This function will clear all cached *meteo* files.

See Also

Other meteo: meteo_show_cache()

meteo_coverage *Determine the "coverage" for a station data frame*

Description

Call this function after pulling down observations for a set of stations to retrieve the "coverage" (i.e. how complete each field is). If either or both obs_start_date or obs_end_date are specified, the coverage test will be limited to that date range.

Usage

```
meteo_coverage(
  meteo_df,
  obs_start_date = NULL,
  obs_end_date = NULL,
  verbose = FALSE
)
```

Arguments

meteo_df	a *meteo* data.frame
obs_start_date	specify either or both (obs_start_date, obs_end_date) to constrain coverate tests. These should be Date objects.
obs_end_date	specify either or both (obs_start_date, obs_end_date) to constrain coverate tests. These should be Date objects.
verbose	if TRUE will display the coverage summary along with returning the coverage data.frame

Details

There is an autoplot method for the output of this function.

Value

a list containing 2 data.frames named 'summary' and 'detail'. The 'summary' data.frame contains columns:

```
$ id         (chr)
$ start_date (time)
$ end_date   (time)
$ total_obs  (int)
```

with additional fields (and their coverage percent) depending on which weather variables were queried and available for the weather station. The data.frame named 'detail' contains the same columns as the meteo_df input data, but expands the rows to contain NAs for days without data.

Examples

```
## Not run:

monitors <- c("ASN00095063", "ASN00024025", "ASN00040112", "ASN00041023",
              "ASN00009998", "ASN00066078", "ASN00003069", "ASN00090162",
              "ASN00040126", "ASN00058161")
obs <- meteo_pull_monitors(monitors)
obs_covr <- meteo_coverage(obs)

if (interactive()) {
  library("ggplot2")
  autoplot(obs_covr)
}

## End(Not run)
```

`meteo_distance` *Find all monitors within a radius of a location*

Description

This function will identify all weather stations with a specified radius of a location. If no radius is given, the function will return a dataframe of all available monitors, sorted by distance to the location. The `limit` argument can be used to limit the output dataframe to the `x` closest monitors to the location.

Usage

```
meteo_distance(
  station_data,
  lat,
  long,
  units = "deg",
  radius = NULL,
  limit = NULL
)
```

Arguments

`station_data`	The output of `ghcnd_stations()`, which is a current list of weather stations available through NOAA for the GHCND dataset. The format of this is a dataframe with one row per weather station. Latitude and longitude for the station locations should be in columns with the names "latitude" and "longitude", consistent with the output from `ghcnd_stations()`. To save time, run the `ghcnd_stations` call and save the output to an object, rather than rerunning the default every time (see the examples in `meteo_nearby_stations()`).
`lat`	Latitude of the location. Southern latitudes should be given as negative values.
`long`	Longitude of the location. Western longitudes should be given as negative values.
`units`	Units of the latitude and longitude values. Possible values are: • `deg`: Degrees (default); • `rad`: Radians.
`radius`	A numeric vector giving the radius (in kilometers) within which to search for monitors near a location.
`limit`	An integer giving the maximum number of monitors to include for each location. The `x` closest monitors will be kept. Default is NULL (pull everything available, within the radius if the radius is specified).

Value

A dataframe of weather stations near the location. This is the single-location version of the return value for `meteo_nearby_stations()`

Author(s)

Alex Simmons <a2.simmons@qut.edu.au>, Brooke Anderson <brooke.anderson@colostate.edu>

Examples

```
## Not run:
station_data <- ghcnd_stations()
meteo_distance(station_data, -33, 151, radius = 10, limit = 10)
meteo_distance(station_data, -33, 151, radius = 10, limit = 3)

# FIXME - units param is ignored
#meteo_distance(station_data, -33, 151, units = 'rad', radius = 10, limit = 3)

## End(Not run)
```

---

meteo_nearby_stations *Find weather monitors near locations*

Description

This function inputs a dataframe with latitudes and longitudes of locations and creates a dataframe with monitors within a certain radius of those locations. The function can also be used, with the `limit` argument, to pull a certain number of the closest weather monitors to each location. The weather monitor IDs in the output dataframe can be used with other **rnoaa** functions to pull data from all available weather stations near a location (e.g., `meteo_pull_monitors()`).

Usage

```
meteo_nearby_stations(
  lat_lon_df,
  lat_colname = "latitude",
  lon_colname = "longitude",
  station_data = ghcnd_stations(),
  var = "all",
  year_min = NULL,
  year_max = NULL,
  radius = NULL,
  limit = NULL
)
```

Arguments

`lat_lon_df`	A dataframe that contains the latitude, longitude, and a unique identifier for each location (`id`). For an example of the proper format for this dataframe, see the examples below. Latitude and longitude must both be in units of decimal degrees. Southern latitudes and Western longitudes should be given as negative values. A tibble is accepted, but is coerced to a data.frame internally before any usage.

`lat_colname` A character string giving the name of the latitude column in the `lat_lon_df` dataframe.

`lon_colname` A character string giving the name of the longitude column in the `lat_lon_df` dataframe.

`station_data` The output of `ghcnd_stations()`, which is a current list of weather stations available through NOAA for the GHCND dataset. The format of this is a dataframe with one row per weather station. Latitude and longitude for the station locations should be in columns with the names "latitude" and "longitude", consistent with the output from `ghcnd_stations()`. To save time, run the `ghcnd_stations` call and save the output to an object, rather than rerunning the default every time (see the examples in `meteo_nearby_stations()`).

`var` A character vector specifying either `"all"` (pull all available weather parameters for the site) or the weather parameters to keep in the final data (e.g., `c("TMAX","TMIN")` to only keep maximum and minimum temperature). Example choices for this argument include:

- `PRCP`: Precipitation, in tenths of millimeters
- `TAVG`: Average temperature, in tenths of degrees Celsius
- `TMAX`: Maximum temperature, in tenths of degrees Celsius
- `TMIN`: Minimum temperature, in tenths of degrees Celsius

A full list of possible weather variables is available in NOAA's README file for the GHCND data . Most weather stations will only have a small subset of all the possible weather variables, so the data generated by this function may not include all of the variables the user specifies through this argument.

`year_min` A numeric value giving the earliest year from which you ultimately want weather data (e.g., 2013, if you only are interested in data from 2013 and later).

`year_max` A numeric value giving the latest year from which you ultimately want weather data.

`radius` A numeric vector giving the radius (in kilometers) within which to search for monitors near a location.

`limit` An integer giving the maximum number of monitors to include for each location. The `x` closest monitors will be kept. Default is NULL (pull everything available, within the radius if the radius is specified).

Details

Great circle distance is used to determine whether a weather monitor is within the required radius.

Value

A list containing dataframes with the sets of unique weather stations within the search radius for each location. Site IDs for the weather stations given in this dataframe can be used in conjunction with other functions in the **rnoaa** package to pull weather data for the station. The dataframe for each location includes:

- `id`: The weather station ID, which can be used in other functions to pull weather data from the station;

- `name`: The weather station name;
- `latitude`: The station's latitude, in decimal degrees. Southern latitudes will be negative;
- `longitude`: The station's longitude, in decimal degrees. Western longitudes will be negative;
- `distance`: The station's distance, in kilometers, from the location.

Note

By default, this function will pull the full station list from NOAA to use to identify nearby locations. If you will be creating lists of monitors nearby several stations, you can save some time by using the `ghcnd_stations()` function separately to create an object with all stations and then use the argument `station_data` in this function to reference that object, rather than using this function's defaults (see examples).

Author(s)

Alex Simmons <a2.simmons@qut.edu.au>, Brooke Anderson <brooke.anderson@colostate.edu>

See Also

The weather monitor IDs generated by this function can be used in other functions in the **rnoaa** package, like `meteo_pull_monitors()` and `meteo_tidy_ghcnd()`, to pull weather data from weather monitors near a location.

Examples

```
## Not run:

station_data <- ghcnd_stations() # Takes a while to run

lat_lon_df <- data.frame(id = c("sydney", "brisbane"),
                         latitude = c(-33.8675, -27.4710),
                         longitude = c(151.2070, 153.0234))
nearby_stations <-  meteo_nearby_stations(lat_lon_df = lat_lon_df,
                    station_data = station_data, radius = 10)

miami <- data.frame(id = "miami", latitude = 25.7617, longitude = -80.1918)

# Get all stations within 50 kilometers
meteo_nearby_stations(lat_lon_df = miami, station_data = station_data,
                      radius = 50, var = c("PRCP", "TMAX"),
                      year_min = 1992, year_max = 1992)
# Get the closest 10 monitors
meteo_nearby_stations(lat_lon_df = miami, station_data = station_data,
                      limit = 10, var = c("PRCP", "TMAX"),
                      year_min = 1992, year_max = 1992)

## End(Not run)
```

`meteo_process_geographic_data` *Calculate the distances between a location and all available stations*

Description

This function takes a single location and a dataset of available weather stations and calculates the distance between the location and each of the stations, using the great circle method. A new column is added to the dataset of available weather stations giving the distance between each station and the input location. The station dataset is then sorted from closest to furthest distance to the location and returned as the function output.

Usage

```
meteo_process_geographic_data(station_data, lat, long, units = "deg")
```

Arguments

`station_data`	The output of `ghcnd_stations()`, which is a current list of weather stations available through NOAA for the GHCND dataset. The format of this is a dataframe with one row per weather station. Latitude and longitude for the station locations should be in columns with the names "latitude" and "longitude", consistent with the output from `ghcnd_stations()`. To save time, run the `ghcnd_stations` call and save the output to an object, rather than rerunning the default every time (see the examples in `meteo_nearby_stations()`).
`lat`	Latitude of the location. Southern latitudes should be given as negative values.
`long`	Longitude of the location. Western longitudes should be given as negative values.
`units`	Units of the latitude and longitude values. Possible values are: • `deg`: Degrees (default); • `rad`: Radians.

Value

The `station_data` dataframe that is input, but with a `distance` column added that gives the distance to the location (in kilometers), and re-ordered by distance between each station and the location (closest weather stations first).

Author(s)

Alex Simmons <a2.simmons@qut.edu.au>, Brooke Anderson <brooke.anderson@colostate.edu>

Examples

```
## Not run:
station_data <- ghcnd_stations()
meteo_process_geographic_data(station_data, lat=-33, long=151)

## End(Not run)
```

`meteo_pull_monitors` *Pull GHCND weather data for multiple weather monitors*

Description

This function takes a vector of one or more weather station IDs. It will pull the weather data from the Global Historical Climatology Network's daily data (GHCND) for each of the stations and join them together in a single tidy dataframe. For any weather stations that the user calls that are not available by ftp from GHCND, the function will return a warning giving the station ID.

Usage

```
meteo_pull_monitors(
  monitors,
  keep_flags = FALSE,
  date_min = NULL,
  date_max = NULL,
  var = "all"
)
```

Arguments

`monitors`	A character vector listing the station IDs for all weather stations the user would like to pull. To get a full and current list of stations, the user can use the `ghcnd_stations()` function. To identify stations within a certain radius of a location, the user can use the `meteo_nearby_stations()` function.
`keep_flags`	TRUE / FALSE for whether the user would like to keep all the flags for each weather variable. The default is to not keep the flags (FALSE). See the note below for more information on these flags.
`date_min`	A character string giving the earliest date of the daily weather time series that the user would like in the final output. This character string should be formatted as "yyyy-mm-dd". If not specified, the default is to keep all daily data for the queried weather site from the earliest available date.
`date_max`	A character string giving the latest date of the daily weather time series that the user would like in the final output. This character string should be formatted as "yyyy-mm-dd". If not specified, the default is to keep all daily data for the queried weather site through the most current available date.

var — A character vector specifying either "all" (pull all available weather parameters for the site) or the weather parameters to keep in the final data (e.g., c("TMAX","TMIN") to only keep maximum and minimum temperature). Example choices for this argument include:

- PRCP: Precipitation, in tenths of millimeters
- TAVG: Average temperature, in tenths of degrees Celsius
- TMAX: Maximum temperature, in tenths of degrees Celsius
- TMIN: Minimum temperature, in tenths of degrees Celsius

A full list of possible weather variables is available in NOAA's README file for the GHCND data . Most weather stations will only have a small subset of all the possible weather variables, so the data generated by this function may not include all of the variables the user specifies through this argument.

Value

A data frame of daily weather data for multiple weather monitors, converted to a tidy format. All weather variables may not exist for all weather stations. Examples of variables returned are:

- id: Character string with the weather station site id
- date: Date of the observation
- prcp: Precipitation, in tenths of mm
- tavg: Average temperature, in tenths of degrees Celsius
- tmax: Maximum temperature, in tenths of degrees Celsius
- tmin: Minimum temperature, in tenths of degrees Celsius
- awnd: Average daily wind speed, in meters / second
- wsfg: Peak gust wind speed, in meters / second

There are other possible weather variables in the Global Historical Climatology Network; see for a full list. If the var argument is something other than "all", then only variables included in that argument will be included in the output data frame. All variables are in the units specified in the linked file (note that, in many cases, measurements are given in tenths of the units more often used, e.g., tenths of degrees for temperature). All column names correspond to variable names in the linked file, but with all uppercase letters changed to lowercase.

Note

The weather flags, which are kept by specifying keep_flags = TRUE are:

- *_mflag: Measurement flag, which gives some information on how the observation was measured.
- *_qflag: Quality flag, which gives quality information on the measurement, like if it failed to pass certain quality checks.
- *_sflag: Source flag. This gives some information on the weather collection system (e.g., U.S. Cooperative Summary of the Day, Australian Bureau of Meteorology) the weather observation comes from.

More information on the interpretation of these flags can be found in the README file for the NCDC's Daily Global Historical Climatology Network's data at

This function converts any value of -9999 to a missing value for the variables "prcp", "tmax", "tmin", "tavg", "snow", and "snwd". However, for some weather observations, there still may be missing values coded using a series of "9"s of some length. You will want to check your final data to see if there are lurking missing values given with series of "9"s.

This function may take a while to run.

Author(s)

Brooke Anderson <brooke.anderson@colostate.edu>

References

For more information about the data pulled with this function, see:

Menne, M.J., I. Durre, R.S. Vose, B.E. Gleason, and T.G. Houston, 2012: An overview of the Global Historical Climatology Network-Daily Database. Journal of Atmospheric and Oceanic Technology, 29, 897-910, doi:10.1175/JTECH-D-11-00103.1.

Examples

```
## Not run:

monitors <- c("ASN00003003", "ASM00094299", "ASM00094995", "ASM00094998")
all_monitors_clean <- meteo_pull_monitors(monitors)


## End(Not run)
```

---

meteo_show_cache *Show the* meteo *cache directory*

Description

Displays the full path to the meteo cache directory

Usage

```
meteo_show_cache()
```

See Also

Other meteo: meteo_clear_cache()

`meteo_spherical_distance` *Calculate the distance between two locations*

Description

This function uses the haversine formula to calculate the great circle distance between two locations, identified by their latitudes and longitudes.

Usage

```
meteo_spherical_distance(lat1, long1, lat2, long2, units = "deg")
```

Arguments

`lat1`	Latitude of the first location.
`long1`	Longitude of the first location.
`lat2`	Latitude of the second location.
`long2`	Longitude of the second location.
`units`	Units of the latitude and longitude values. Possible values are: • `deg`: Degrees (default); • `rad`: Radians.

Value

A numeric value giving the distance (in kilometers) between the pair of locations.

Note

This function assumes an earth radius of 6,371 km.

Author(s)

Alex Simmons <a2.simmons@qut.edu.au>, Brooke Anderson <brooke.anderson@colostate.edu>

Examples

```
meteo_spherical_distance(lat1 = -27.4667, long1 = 153.0217,
                         lat2 = -27.4710, long2 = 153.0234)
```

`meteo_tidy_ghcnd` *Create a tidy GHCND dataset from a single monitor*

Description

This function inputs an object created by ghcnd and cleans up the data into a tidy form.

Usage

```
meteo_tidy_ghcnd(
  stationid,
  keep_flags = FALSE,
  var = "all",
  date_min = NULL,
  date_max = NULL
)
```

Arguments

`stationid`	(character) A character vector giving the identification of the weather stations for which the user would like to pull data. To get a full and current list of stations, the user can use the `ghcnd_stations()` function. To identify stations within a certain radius of a location, the user can use the `meteo_nearby_stations()` function.
`keep_flags`	TRUE / FALSE for whether the user would like to keep all the flags for each weather variable. The default is to not keep the flags (FALSE). See the note below for more information on these flags.
`var`	A character vector specifying either `"all"` (pull all available weather parameters for the site) or the weather parameters to keep in the final data (e.g., `c("TMAX","TMIN")` to only keep maximum and minimum temperature). Example choices for this argument include: • `PRCP`: Precipitation, in tenths of millimeters • `TAVG`: Average temperature, in tenths of degrees Celsius • `TMAX`: Maximum temperature, in tenths of degrees Celsius • `TMIN`: Minimum temperature, in tenths of degrees Celsius A full list of possible weather variables is available in NOAA's README file for the GHCND data . Most weather stations will only have a small subset of all the possible weather variables, so the data generated by this function may not include all of the variables the user specifies through this argument.
`date_min`	A character string giving the earliest date of the daily weather time series that the user would like in the final output. This character string should be formatted as "yyyy-mm-dd". If not specified, the default is to keep all daily data for the queried weather site from the earliest available date.

`date_max` A character string giving the latest date of the daily weather time series that the user would like in the final output. This character string should be formatted as "yyyy-mm-dd". If not specified, the default is to keep all daily data for the queried weather site through the most current available date.

Value

A data frame of daily weather data for a single weather monitor, converted to a tidy format. All weather variables may not exist for all weather stations. Examples of variables returned are:

- `id`: Character string with the weather station site id
- `date`: Date of the observation
- `prcp`: Precipitation, in tenths of mm
- `tavg`: Average temperature, in degrees Celsius
- `tmax`: Maximum temperature, in degrees Celsius
- `tmin`: Minimum temperature, in degrees Celsius
- `awnd`: Average daily wind speed, in meters / second
- `wsfg`: Peak gust wind speed, in meters / second

There are other possible weather variables in the Global Historical Climatology Network; see for a full list. The variables `prcp`, `tmax`, `tmin`, and `tavg` have all been converted from tenths of their metric to the metric (e.g., from tenths of degrees Celsius to degrees Celsius). All other variables are in the units specified in the linked file.

Note

The weather flags, which are kept by specifying `keep_flags = TRUE` are:

- *_mflag: Measurement flag, which gives some information on how the observation was measured.
- *_qflag: Quality flag, which gives quality information on the measurement, like if it failed to pass certain quality checks.
- *_sflag: Source flag. This gives some information on the weather collection system (e.g., U.S. Cooperative Summary of the Day, Australian Bureau of Meteorology) the weather observation comes from.

More information on the interpretation of these flags can be found in the README file for the NCDC's Daily Global Historical Climatology Network's data at

Author(s)

Brooke Anderson <brooke.anderson@colostate.edu>

See Also

`meteo_pull_monitors()`

Examples

```
## Not run:
# One station in Australia is ASM00094275
meteo_tidy_ghcnd(stationid = "ASN00003003")
meteo_tidy_ghcnd(stationid = "ASN00003003", var = "tavg")
meteo_tidy_ghcnd(stationid = "ASN00003003", date_min = "1989-01-01")

## End(Not run)
```

`meteo_tidy_ghcnd_element`

Restructure element of ghcnd_search list

Description

This function restructures the output of `ghcnd_search()` to add a column giving the variable name (`key`) and change the name of the variable column to `value`. These changes facilitate combining all elements from the list created by `ghcnd_search()`, to create a tidy dataframe of the weather observations from the station.

Usage

```
meteo_tidy_ghcnd_element(x, keep_flags = FALSE)
```

Arguments

`x`	A dataframe with daily observations for a single monitor for a single weather variable. This dataframe is one of the elements returned by `ghcnd_search()`
`keep_flags`	TRUE / FALSE for whether the user would like to keep all the flags for each weather variable. The default is to not keep the flags (FALSE). See the note below for more information on these flags.

Value

A dataframe reformatted to allow easy aggregation of all weather variables for a single monitor.

Author(s)

Brooke Anderson <brooke.anderson@colostate.edu>

ncdc *Search for and get NOAA NCDC data*

Description

Search for and get NOAA NCDC data

Usage

```
ncdc(
  datasetid = NULL,
  datatypeid = NULL,
  stationid = NULL,
  locationid = NULL,
  startdate = NULL,
  enddate = NULL,
  sortfield = NULL,
  sortorder = NULL,
  limit = 25,
  offset = NULL,
  token = NULL,
  includemetadata = TRUE,
  add_units = FALSE,
  ...
)
```

Arguments

datasetid	(required) Accepts a single valid dataset id. Data returned will be from the dataset specified, see ncdc_datasets
datatypeid	Accepts a valid data type id or a vector or list of data type ids. (optional)
stationid	Accepts a valid station id or a vector or list of station ids
locationid	Accepts a valid location id or a vector or list of location ids (optional)
startdate	(character/date) 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. required.
enddate	(character/date) 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. required.
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) See **Authentication** section below for more details.
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.
add_units	(logical) whether to add units information or not. default: FALSE. If TRUE, after getting data from NOAA we add a new column units. See "Adding units" in Details for more
...	Curl options passed on to HttpClient (optional)

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.

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. Note that values in the data slot don't indicate their units by default, so you will want to either use the add_units parameter (experimental, see Adding units) or consult the documentation for each dataset to ensure you're using the correct units.

Authentication

Get an API key (aka, token) at You can pass your token in as an argument or store it one of two places:

- your .Rprofile file with the entry `options(noaakey = "your-noaa-token")`
- your .Renviron file with the entry `NOAA_KEY=your-noaa-token`

See Startup for information on how to create/find your .Rrofile and .Renviron files

Flags

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.

- fl_c completeness
- fl_d day
- fl_m measurement
- fl_q quality
- fl_s source
- fl_t time
- fl_cmiss consecutive missing
- fl_miss missing
- fl_u units

GSOM/GSOY Flags

Note that flags are different for GSOM and GSOY datasets. They have their own set of flags per data class. See `system.file("extdata/gsom.json",package = "rnoaa")` for GSOM and `system.file("extdata/gsom.json",package = "rnoaa")` for GSOY. Those are JSON files. The `system.file()` call gives you then path, then read in with `jsonlite::fromJSON()` which will give a data.frame of the metadata. For more detailed info but plain text, open `system.file("extdata/gsom_readme.txt",packa` `= "rnoaa")` and `system.file("extdata/gsoy_readme.txt",package = "rnoaa")` in a text editor.

Adding units

The `add_units` parameter is experimental - USE WITH CAUTION! If `add_units=TRUE` we pull data from curated lists of data used by matching by datasetid and data type.

We've attempted to gather as much information as possible on the many, many data types across the many different NOAA data sets. However, we may have got some things wrong, so make sure to double check data you get if you do add units.

Get in touch if you find some units that are wrong or missing, and if you are able to help correct information.

See Also

Other ncdc: `ncdc_combine()`, `ncdc_datacats()`, `ncdc_datasets()`, `ncdc_datatypes()`, `ncdc_locs_cats()`, `ncdc_locs()`, `ncdc_plot()`, `ncdc_stations()`

Examples

```
## 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')
### also accepts dates as class Date
ncdc(datasetid='GHCND', stationid='GHCND:USW00014895',
   startdate = as.Date('2013-10-01'), enddate = as.Date('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 GSOM) data from October 1 2013 to December 1 2013
ncdc(datasetid='GSOM', startdate = '2013-10-01', enddate = '2013-12-01')
ncdc(datasetid='GSOM', startdate = '2013-10-01', enddate = '2013-12-01',
   stationid = "GHCND:AE000041196")

# 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')

# multiple datatypeid's
ncdc(datasetid='PRECIP_HLY', datatypeid = 'HPCP',
   startdate = '2010-05-01', enddate = '2010-05-10')

# multiple locationid's
ncdc(datasetid='PRECIP_HLY', locationid=c("FIPS:30103", "FIPS:30091"),
   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')

# 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 GSOY 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')
ncdc(datasetid='GHCND', stationid='GHCND:USW00014895',
   startdate = '2013-10-01', enddate = '2013-12-01', includemetadata=FALSE)

# Many stationid's
stat <- ncdc_stations(startdate = "2000-01-01", enddate = "2016-01-01")
## find out what datasets might be available for these stations
ncdc_datasets(stationid = stat$data$id[10])
## get some data
ncdc(datasetid = "GSOY", stationid = stat$data$id[1:10],
   startdate = "2010-01-01", enddate = "2011-01-01")

## 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

See Also

Other ncdc: ncdc_datacats(), ncdc_datasets(), ncdc_datatypes(), ncdc_locs_cats(), ncdc_locs(), ncdc_plot(), ncdc_stations(), ncdc()

Examples

```
## Not run:
# data
out1 <- ncdc(datasetid='GHCND', locationid = 'FIPS:02', startdate = '2010-05-01',
enddate = '2010-05-31', limit=10)
out2 <- ncdc(datasetid='GHCND', 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='GHCND', locationid='FIPS:12017',
stationid='GHCND:USC00084289')
out2 <- ncdc_stations(stationid='COOP:010008')
out3 <- ncdc_stations(datasetid='PRECIP_HLY', startdate='19900101',
```

```
enddate='19901231')
out4 <- ncdc_stations(datasetid='GHCND', locationid='FIPS:12017')
ncdc_combine(out1, out2, out3, out4)

# try to combine two different classes
out1 <- ncdc_locs_cats(startdate='1970-01-01')
out2 <- ncdc_stations(stationid='COOP:010008')
out3 <- ncdc_locs_cats(locationcategoryid='CLIM_REG')
ncdc_combine(out1, out2, out3)

## End(Not run)
```

`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 valid dataset id or a vector or list of dataset id's. 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 or a vector or list of station ids (optional)
`locationid`	Accepts a valid location id or a vector or list of location id's. (optional)
`startdate`	Accepts 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. Paramater 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) See **Authentication** section below for more details.
...	Curl options passed on to `HttpClient`

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.

Authentication

Get an API key (aka, token) at You can pass your token in as an argument or store it one of two places:

- your .Rprofile file with the entry `options(noaakey = "your-noaa-token")`
- your .Renviron file with the entry `NOAA_KEY=your-noaa-token`

See `Startup` for information on how to create/find your .Rrofile and .Renviron files

See Also

Other ncdc: `ncdc_combine()`, `ncdc_datasets()`, `ncdc_datatypes()`, `ncdc_locs_cats()`, `ncdc_locs()`, `ncdc_plot()`, `ncdc_stations()`, `ncdc()`

Examples

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

## by datasetid
ncdc_datacats(datasetid="ANNUAL")
ncdc_datacats(datasetid=c("ANNUAL", "PRECIP_HLY"))

## Single data category
```

```
ncdc_datacats(datacategoryid="ANNAGR")

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

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

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

## Curl debugging
ncdc_datacats(limit=10, verbose = TRUE)

## End(Not run)
```

`ncdc_datasets` *Search NOAA datasets*

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

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

Arguments

`datasetid`	(optional) Accepts a single valid dataset id. Data returned will be from the dataset specified.

datatypeid	Accepts a valid data type id or a vector or list of data type ids. (optional)
stationid	Accepts a valid station id or a vector or list of station ids
locationid	Accepts a valid location id or a vector or list of location ids (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) See **Authentication** section below for more details.
...	Curl options passed on to `HttpClient` (optional)

Value

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

Authentication

Get an API key (aka, token) at You can pass your token in as an argument or store it one of two places:

- your .Rprofile file with the entry `options(noaakey = "your-noaa-token")`
- your .Renviron file with the entry `NOAA_KEY=your-noaa-token`

See `Startup` for information on how to create/find your .Rrofile and .Renviron files

See Also

Other ncdc: `ncdc_combine()`, `ncdc_datacats()`, `ncdc_datatypes()`, `ncdc_locs_cats()`, `ncdc_locs()`, `ncdc_plot()`, `ncdc_stations()`, `ncdc()`

Examples

```
## 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')
## two datatypeid's
ncdc_datasets(datatypeid=c('TOBS', "ACMH"))

# Get datasets with data for a series of the same parameter arg, in this case
# stationid's
ncdc_datasets(stationid='COOP:310090')
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'))
ncdc_datasets(datasetid='GSOY', datatypeid=c('ACMC','ACMH','ACSC'))

# Multiple locationid's
ncdc_datasets(locationid="FIPS:30091")
ncdc_datasets(locationid=c("FIPS:30103", "FIPS:30091"))

## 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. For example: If it's 64 degrees out right now, then the data type is Air Temperature and the data is 64.

Usage

```
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,
  ...
)
```

Arguments

datasetid	(optional) Accepts a valid dataset id or a vector or list of them. Data returned will be from the dataset specified.
datatypeid	Accepts a valid data type id or a vector or list of data type ids. (optional)
datacategoryid	Optional. Accepts a valid data category id or a vector or list of data category ids (although it is rare to have a data type with more than one data category)
stationid	Accepts a valid station id or a vector or list of station ids
locationid	Accepts a valid location id or a vector or list of location ids (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) See **Authentication** section below for more details.
...	Curl options passed on to `HttpClient` (optional)

Value

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

Authentication

Get an API key (aka, token) at You can pass your token in as an argument or store it one of two places:

- your .Rprofile file with the entry `options(noaakey = "your-noaa-token")`
- your .Renviron file with the entry `NOAA_KEY=your-noaa-token`

See `Startup` for information on how to create/find your .Rrofile and .Renviron files

See Also

Other ncdc: `ncdc_combine()`, `ncdc_datacats()`, `ncdc_datasets()`, `ncdc_locs_cats()`, `ncdc_locs()`, `ncdc_plot()`, `ncdc_stations()`, `ncdc()`

Examples

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

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

# datasetid, one or many
## ANNUAL should be replaced by GSOY, but both exist and give
## different answers
ncdc_datatypes(datasetid="ANNUAL")
ncdc_datatypes(datasetid="GSOY")
ncdc_datatypes(datasetid=c("ANNUAL", "PRECIP_HLY"))

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

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

# Fetch data types that support a given set of loncationids
ncdc_datatypes(locationid='CITY:AG000001')
ncdc_datatypes(locationid=c('CITY:AG000001','CITY:AG000004'))

## End(Not run)
```

---

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.

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 valid dataset id or a vector or list of dataset id's. Data returned will be from the dataset specified, see datasets() (required)
locationid	A valid location id or a vector or list of location ids.
locationcategoryid	A valid location id or a vector or list of location category ids
startdate	A valid ISO formatted 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 formatted date (yyyy-mm-dd). Data returned will have data before the specified date. Paramater 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) See **Authentication** section below for more details.
...	Curl options passed on to `HttpClient`

Value

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

Authentication

Get an API key (aka, token) at You can pass your token in as an argument or store it one of two places:

- your .Rprofile file with the entry `options(noaakey = "your-noaa-token")`
- your .Renviron file with the entry `NOAA_KEY=your-noaa-token`

See `Startup` for information on how to create/find your .Rrofile and .Renviron files

See Also

Other ncdc: `ncdc_combine()`, `ncdc_datacats()`, `ncdc_datasets()`, `ncdc_datatypes()`, `ncdc_locs_cats()`, `ncdc_plot()`, `ncdc_stations()`, `ncdc()`

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')
ncdc_locs(datasetid=c('GHCND', 'ANNUAL'))
ncdc_locs(datasetid=c('GSOY', 'ANNUAL'))
ncdc_locs(datasetid=c('GHCND', 'GSOM'))

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

# Many locationcategoryid's
## this apparently works, but returns nothing often with multiple
## locationcategoryid's
ncdc_locs(locationcategoryid=c('ST', 'ZIP'))

# 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 valid dataset id or a vector or list of dataset id's. Data returned will be from the dataset specified, see datasets() (required)
locationcategoryid	A valid location id or a vector or list of location category ids
startdate	A valid ISO formatted 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 formatted date (yyyy-mm-dd). Data returned will have data before the specified date. Paramater 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) See **Authentication** section below for more details.
...	Curl options passed on to `HttpClient`

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.

Authentication

Get an API key (aka, token) at You can pass your token in as an argument or store it one of two places:

- your .Rprofile file with the entry `options(noaakey = "your-noaa-token")`
- your .Renviron file with the entry `NOAA_KEY=your-noaa-token`

See `Startup` for information on how to create/find your .Rrofile and .Renviron files

See Also

Other ncdc: `ncdc_combine()`, `ncdc_datacats()`, `ncdc_datasets()`, `ncdc_datatypes()`, `ncdc_locs()`, `ncdc_plot()`, `ncdc_stations()`, `ncdc()`

Examples

```
## 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')
ncdc_locs_cats(datasetid='GSOY')
ncdc_locs_cats(datasetid='ANNUAL')

# multiple datasetid's
ncdc_locs_cats(datasetid=c('GHCND', 'GSOM'))

# 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

Plot NOAA climate data.

Usage

```
ncdc_plot(..., breaks = NULL, dateformat = "%d/%m/%y")
```

Arguments

...	Input noaa object or objects.
breaks	Regularly spaced date breaks for x-axis. See examples for usage. See date_breaks. Default: NULL (uses ggplot2 default break sformatting)
dateformat	Date format using standard POSIX specification for labels on x-axis. See date_format()

Details

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

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. Look at the internals for what the function does.

Value

ggplot2 plot

See Also

Other ncdc: ncdc_combine(), ncdc_datacats(), ncdc_datasets(), ncdc_datatypes(), ncdc_locs_cats(), ncdc_locs(), ncdc_stations(), ncdc()

Examples

```
## Not run:
# Search for data first, then plot
out <- ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', datatypeid='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")

# Combine many calls to ncdc function
out1 <- ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', datatypeid='PRCP',
   startdate = '2010-03-01', enddate = '2010-05-31', limit=500)
out2 <- ncdc(datasetid='GHCND', stationid='GHCND:USW00014895', datatypeid='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)
```

---

ncdc_stations *Get metadata about NOAA NCDC stations.*

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,
  token = NULL,
  dataset = NULL,
  station = NULL,
  location = NULL,
  locationtype = NULL,
  page = NULL,
  ...
)
```

Arguments

stationid	A single valid station id, with datasetid namespace, e.g., GHCND:USW00014895
datasetid	(optional) Accepts a valid dataset id or a vector or list of them. Data returned will be from the dataset specified.
datatypeid	Accepts a valid data type id or a vector or list of data type ids. (optional)
locationid	Accepts a valid location id or a vector or list of location ids (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 a vector or list of data category ids.
extent	(numeric, optional) The geographical extent for which you want to search. 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)`.
token	This must be a valid token token supplied to you by NCDC's Climate Data Online access token generator. (required) See **Authentication** section below for more details.
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.
...	Curl options passed on to HttpClient (optional)

Value

A list of metadata.

Authentication

Get an API key (aka, token) at You can pass your token in as an argument or store it one of two places:

- your .Rprofile file with the entry `options(noaakey = "your-noaa-token")`
- your .Renviron file with the entry `NOAA_KEY=your-noaa-token`

See Startup for information on how to create/find your .Rrofile and .Renviron files

See Also

Other ncdc: ncdc_combine(), ncdc_datacats(), ncdc_datasets(), ncdc_datatypes(), ncdc_locs_cats(), ncdc_locs(), ncdc_plot(), ncdc()

Examples

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

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

# For many stations use lapply or similar
lapply(c("COOP:010008", "COOP:010063", "COOP:010116"), function(z) {
  ncdc_stations(
   startdate = "2013-01-01",
   enddate = "2014-11-01",
   stationid = z)
}$data)

# Displays all stations within GHCN-Daily (100 Stations per page limit)
ncdc_stations(datasetid = 'GHCND')
ncdc_stations(datasetid = 'ANNUAL')
ncdc_stations(datasetid = 'GSOY')

# Station
```

```
ncdc_stations(datasetid='NORMAL_DLY', stationid='GHCND:USW00014895')

# datatypeid
ncdc_stations(datatypeid="ANN-HTDD-NORMAL")
ncdc_stations(datatypeid=c("ANN-HTDD-NORMAL", "ACSC"))

# locationid
ncdc_stations(locationid="CITY:AG000001")
ncdc_stations(locationid="FIPS:30091")
ncdc_stations(locationid=c("FIPS:30103", "FIPS:30091"))

# datacategoryid
ncdc_stations(datacategoryid="ANNPRCP")
ncdc_stations(datacategoryid="AUAGR")
ncdc_stations(datacategoryid=c("ANNPRCP", "AUAGR"))

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

# Specify datasetid and locationid
ncdc_stations(datasetid='GHCND', locationid='FIPS:12017')

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

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

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

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

# Search for stations by spatial extent
## Search using a bounding box, w/ lat/long of the SW corner, then of NE corner
ncdc_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 package rerddap
- erddap_clear_cache: See package rerddap
- erddap_datasets: Moved to package rerddap
- erddap_grid: Moved to package rerddap
- erddap_info: Moved to rerddap::info()
- erddap_search: Moved to rerddap::ed_search
- erddap_table: Moved to rerddap::tabledap
- ncdc_leg_variables: Removed. See NCDC Legacy below
- ncdc_leg_sites: Removed. See NCDC Legacy below
- ncdc_leg_site_info: Removed. See NCDC Legacy below
- ncdc_leg_data: Removed. See NCDC Legacy below
- seaice: Replaced with sea_ice()
- lcd_cleanup: No longer available. See lcd docs
- ghcnd_clear_cache: No longer available. See rnoaa_caching
- storm_shp: Function defunct.
- storm_shp_read: Function defunct.
- storm_data: Function defunct.
- storm_meta: Function defunct.

Details

The functions for working with GEFS ensemble forecast data (prefixed with "gefs") are defunct, but may come back to rnoaa later:

- gefs()
- gefs_dimension_values()
- gefs_dimensions()
- gefs_ensembles()
- gefs_latitudes()
- gefs_longitudes()
- gefs_times()
- gefs_variables()

NCDC Legacy

The NCDC legacy API is too unreliable and slow. Use the newer NCDC API via the functions ncdc(), ncdc_datacats(), ncdc_datasets(), ncdc_datatypes(), ncdc_locs(), ncdc_locs_cats(), ncdc_stations(), ncdc_plot(), and ncdc_combine()

rnoaa_caching *rnoaa caching*

Description

Manage data caches

Details

To get the cache directory for a data source, see the method `x$cache_path_get()`

`cache_delete` only accepts 1 file name, while `cache_delete_all` doesn't accept any names, but deletes all files. For deleting many specific files, use `cache_delete` in a `lapply()` type call

Note that cached files will continue to be used until they are deleted. It's possible to run into problems when changes happen in your R setup. For example, at least one user reported changing versions of this package and running into problems because a cached data file from a previous version of rnoaa did not work with the newer version of rnoaa. You should occassionally delete all cached files.

Useful user functions

Assuming x is a `HoardClient` class object, e.g., `lcd_cache`

- `x$cache_path_get()` get cache path
- `x$cache_path_set()` set cache path
- `x$list()` returns a character vector of full path file names
- `x$files()` returns file objects with metadata
- `x$details()` returns files with details
- `x$delete()` delete specific files
- `x$delete_all()` delete all files, returns nothing

Caching objects for each data source

- `isd()`/`isd_stations()`: `isd_cache`
- `cpc_prcp()`: `cpc_cache`
- `arc2()`: `arc2_cache`
- `lcd()`: `lcd_cache`
- `bsw()`: `bsw_cache`
- `ersst()`: `ersst_cache`
- `tornadoes()`: `torn_cache`
- `ghcnd()`/`ghcnd_search()`: `ghcnd_cache`
- `se_data()`/`se_files()`: `stormevents_cache`

See Also

rnoaa_options() for managing whether you see messages about cached files when you request data

---

rnoaa_options *rnoaa options*

Description

rnoaa options

Usage

```
rnoaa_options(cache_messages = TRUE)
```

Arguments

cache_messages	(logical) whether to emit messages with information on caching status for function calls that can cache data. default: TRUE

Details

rnoaa package level options; stored in an internal package environment roenv

See Also

rnoaa_caching for managing cached files

Examples

```
## Not run:
rnoaa_options(cache_messages = FALSE)

## End(Not run)
```

sea_ice *Get sea ice data.*

Description

Get sea ice data.

Usage

```
sea_ice(year = NULL, month = NULL, pole = NULL, format = "shp", ...)
```

Arguments

year	(numeric) a year
month	(character) a month, as character abbrevation of a month
pole	(character) one of S (south) or N (north)
format	(character) one of shp (default), geotiff-extent (for geotiff extent data), or geotiff-conc (for geotiff concentration data)
...	Further arguments passed on to `rgdal::readshpfile()` if `format="shp"` or `raster::raster()` if not

Value

data.frame if `format="shp"` (a fortified sp object); `raster::raster()` if not

References

See the "User Guide" pdf at

See Also

`sea_ice_tabular()`

Examples

```
## Not run:
if (requireNamespace("raster")) {

## one year, one moth, one pole
sea_ice(year = 1990, month = "Apr", pole = "N")
sea_ice(year = 1990, month = "Apr", pole = "N", format = "geotiff-extent")
sea_ice(year = 1990, month = "Apr", pole = "N", format = "geotiff-conc")

## one year, one month, many poles
sea_ice(year = 1990, month = "Apr")

## one year, many months, many poles
sea_ice(year = 1990, month = c("Apr", "Jun", "Oct"))
```

```
## many years, one month, one pole
sea_ice(year = 1990:1992, month = "Sep", pole = "N")

# get geotiff instead of shp data.
x <- sea_ice(year = 1990, month = "Apr", format = "geotiff-extent")
y <- sea_ice(year = 1990, month = "Apr", format = "geotiff-conc")
}


## End(Not run)
```

`sea_ice_tabular` *Sea ice tabular data*

Description

Collects `.csv` files from NOAA, and binds them together into a single data.frame. Data across years, with extent and area of ice.

Usage

```
sea_ice_tabular(...)
```

Arguments

`...`	Curl options passed on to crul::verb-GET - beware that curl options are passed to each http request, for each of 24 requests.

Details

An example file, for January, North pole: ftp://sidads.colorado.edu/DATASETS/NOAA/G02135/north/monthly/data/N_01_e

a value in any cell of -9999 indicates missing data

Value

A data.frame with columns:

- year (integer)
- mo (integer)
- data.type (character)
- region (character)
- extent (numeric)
- area (numeric)

See Also

`sea_ice()`

Examples

```
## Not run:
df <- sea_ice_tabular()
df

## End(Not run)
```

`storm_events` *NOAA Storm Events data*

Description

NOAA Storm Events data

Usage

```
se_data(year, type, overwrite = TRUE, ...)

se_files(...)
```

Arguments

`year`	(numeric) a four digit year. see output of `se_files()` for available years. required.
`type`	(character) one of details, fatalities, locations, or legacy. required.
`overwrite`	(logical) To overwrite the path to store files in or not, Default: `TRUE`
`...`	Curl options passed on to crul::verb-GET (optional)

Value

A tibble (data.frame)

Note

See stormevents_cache for managing cached files

Examples

```
## Not run:
# get list of files and their urls
res <- se_files()
res
tail(res)

# get data
x <- se_data(year = 2013, type = "details")
x
```

```
z <- se_data(year = 1988, type = "fatalities")
z

w <- se_data(year = 2003, type = "locations")
w

leg <- se_data(year = 2003, type = "legacy")
leg

## 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

```
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,
  ...
)
```

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. Time out issues likely to occur at higher limits.
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). BEWARE: As far as we know, this parameter doesn't do anything, or at least does not in fact limit the search to the given radius. DO NOT USE.
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 format=xml or format=csv (optional)
...	Curl options passed on to crul::verb-GET (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. Available to government and military users only. If you aren't one of those, you'll get a 400 status stop message if you request data from this dataset (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

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

# Get all 'nx3tvs' near latitude = 32.7 and longitude = -102.0
swdi(dataset='nx3tvs', startdate='20060506', enddate='20060507',
center=c(-102.0,32.7))

# use an id
swdi(dataset='warn', startdate='20060506', enddate='20060507', id=533623)

# Get all 'plsr' within the bounding box (-91,30,-90,31)
swdi(dataset='plsr', startdate='20060505', enddate='20060510',
bbox=c(-91,30,-90,31))

# Get all 'nx3tvs' within the tile -102.1/32.6 (-102.15,32.55,-102.25,32.65)
swdi(dataset='nx3tvs', startdate='20060506', enddate='20060507',
tile=c(-102.12,32.62))

# Counts
## Note: stat='count' will only return metadata, nothing in the data or shape slots
## Note: stat='tilesum:...' returns counts in the data slot for each date for that tile,
##   and shape data
## Get number of 'nx3tvs' near latitude = 32.7 and longitude = -102.0
swdi(dataset='nx3tvs', startdate='20060505', enddate='20060516',
center=c(-102.0,32.7), stat='count')

## Get daily count nx3tvs features on .1 degree grid centered at latitude = 32.7
## and longitude = -102.0
swdi(dataset='nx3tvs', startdate='20060505', enddate='20090516',
stat='tilesum:-102.0,32.7')

# CSV format
swdi(dataset='nx3tvs', startdate='20060505', enddate='20060506', format='csv')

# SHP format
swdi(dataset='nx3tvs', startdate='20060505', enddate='20060506', format='shp',
   filepath='myfile')

# KMZ format
swdi(dataset='nx3tvs', startdate='20060505', enddate='20060506', format='kmz',
   filepath='myfile.kmz')

# csv output to SpatialPointsDataFrame
res <- swdi(dataset='nx3tvs', startdate='20060505', enddate='20060506', format="csv")
library('sp')
coordinates(res$data) <- ~lon + lat
res$data
class(res$data)

## End(Not run)
```

tornadoes *Get NOAA tornado data.*

Description

This function gets spatial paths of tornadoes from NOAA's National Weather Service Storm Prediction Center Severe Weather GIS web page.

Usage

```
tornadoes(...)
```

Arguments

...	Curl options passed on to crul::verb-GET (optional)

Value

A Spatial object is returned of class SpatialLinesDataFrame.

Note

See torn_cache for managing cached files

Examples

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

## End(Not run)
```

vis_miss *Visualize missingness in a dataframe*

Description

Gives you an at-a-glance ggplot of the missingness inside a dataframe, colouring cells according to missingness, where black indicates a present cell and grey indicates a missing cell. As it returns a `ggplot` object, it is very easy to customize and change labels, and so on.

Usage

```
vis_miss(x, cluster = FALSE, sort_miss = FALSE)
```

Arguments

x	a data.frame
cluster	logical TRUE/FALSE. TRUE specifies that you want to use hierarchical clustering (mcquitty method) to arrange rows according to missingness. FALSE specifies that you want to leave it as is.
sort_miss	logical TRUE/FALSE. TRUE arranges the columns in order of missingness.

Details

vis_miss visualises a data.frame to display missingness. This is taken from the visdat package, currently only available on github:

Examples

```
## Not run:
  monitors <- c("ASN00003003", "ASM00094299")
  weather_df <- meteo_pull_monitors(monitors)
  vis_miss(weather_df)

## End(Not run)
```

Index