Package: preMetabolizer (via r-universe)

June 9, 2026

Title A Utility Package for the streamMetabolizer Package

Version 0.0.0.9000

Description Tools for preparing environmental time series before fitting stream metabolism models. The package includes helpers for downloading meteorological and elevation data, aligning irregular observations, converting units, and calculating light, pressure, dissolved gas, and water property inputs for streamMetabolizer.

License AGPL (>= 3)

URL https://connorb.github.io/preMetabolizer/, https://github.com/ConnorB/preMetabolizer

BugReports https://github.com/ConnorB/preMetabolizer/issues

Depends R (>= 4.1.0)

Imports lubridate, dplyr, geosphere, httr2, memoise, methods, nasapower, readr, rlang, sf, stats, stringr, SunCalcMeeus, tibble, tools, cli, Rcpp, lifecycle

Suggests ggplot2, jsonlite, knitr, quarto, testthat (>= 3.0.0), tidyr, withr

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

LinkingTo Rcpp

Config/roxygen2/version 8.0.0.9000

RoxygenNote 8.0.0.9000

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libicu-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev

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

Date/Publication 2026-06-08 23:54:31 UTC

RemoteUrl https://github.com/ConnorB/preMetabolizer

RemoteRef HEAD

RemoteSha b48a794026989efe6a91a360707c4efbec2e38ba

Contents

`preMetabolizer-package`

preMetabolizer: Prepare data for stream metabolism modeling

Description

preMetabolizer provides helpers for building the environmental time series commonly needed before fitting stream metabolism models. It includes tools to download external meteorological and elevation data, align irregular time series, convert hydrology and atmospheric units, calculate water and gas chemistry quantities, and assemble light and pressure inputs for streamMetabolizer.

Coordinates

Functions that require locations use `latitude` and `longitude` in decimal degrees. Latitude ranges from -90 to 90, longitude ranges from -180 to 180, and western longitudes are negative.

Main workflows

- Find and download meteorological data with `get_noaa_stations()`, `closest_noaa_stations()`, `get_ghcnh()`, `ncei_data()`, `get_nasa_data()`, and `ks_meso_timeseries()`. Retrieve mesonet station observations with `iem_networks()`, `iem_current()`, `iem_obhistory()`, `tex_meso_stations()`, `tex_meso_current()`, and `tex_meso_timeseries()`.
- Retrieve or calculate site context with `get_usgs_elev()`, `correct_bp()`, `convert_to_solar_time()`, and `calc_par()`.
- Prepare model inputs with `even_timesteps()`, `calc_O2sat()`, `calc_water_height()`, and the built-in example datasets.
- Summarize and check data with `flag_z()`, `calc_exceedance_prob()`, `calc_cv()`, and `calc_bin_width()`.

Built-in datasets

french_creek contains 5-minute dissolved oxygen and water temperature observations for a complete metabolism-preparation example. kings_discharge contains daily USGS discharge, gage height, and water temperature observations for flow-duration and seasonal summaries.

Author(s)

Maintainer: Connor Brown <ConnorBrown1996@gmail.com> (ORCID)

Authors:

- Connor Brown <ConnorBrown1996@gmail.com> (ORCID)

See Also

streamMetabolizer::metab(), https://connorb.github.io/preMetabolizer/, and https://github.com/ConnorB/preMetabolizer/issues.

---

calc_bin_width *Calculate histogram bin width*

Description

Computes a suggested bin width for a histogram using one of several classical rules. NA values are silently removed before computation.

Usage

```
calc_bin_width(x, method = "auto")
```

Arguments

x	Numeric vector of observations. Must contain at least two distinct, finite, non-NA values.
method	Single character string specifying the binning rule. One of:

"auto" Minimum of Freedman-Diaconis and Sturges (default). Tends to work well across a wide range of distributions.

"sturges" Sturges' rule: $h = R/(\log_2 n + 1)$. Assumes approximate normality; tends to undersmooth for large n.

"fd" Freedman-Diaconis: $h = 2 \cdot \mathrm{IQR}(x) \cdot n^{-1/3}$. Robust to outliers. Falls back to Sturges when $\mathrm{IQR}(x) = 0$.

"sqrt" Square-root rule: $h = R/\sqrt{n}$. Simple heuristic used by some spreadsheet applications.

"rice" Rice rule: $h = R/(2n^{1/3})$. Similar to Sturges but grows more slowly.

"scott" Scott's rule: $h = 3.49\hat{\sigma}n^{-1/3}$. Optimal for normal data in the sense of minimising mean integrated squared error.

"doane" Doane's rule. Extends Sturges to account for skewness; better suited to non-normal distributions. Requires $n \geq 3$.

Details

In all formulae, R denotes the range (`diff(range(x))`) and n the number of non-NA observations.

The `"fd"` fallback to Sturges when `IQR(x) == 0` avoids a zero-width bin, which can occur with heavily discrete or zero-inflated data.

Value

A single positive numeric value giving the bin width.

Errors

The function aborts with an informative message when:

- `x` is not a numeric vector.
- `method` is not a single character string.
- `x` contains no non-NA values, or only one.
- All non-NA values in `x` are identical (zero range).
- `method = "doane"` is requested with fewer than 3 observations (the skewness standard error is undefined for $n < 3$).
- An unrecognised `method` string is supplied.

Examples

```
# Continuous data — Freedman-Diaconis
calc_bin_width(rnorm(200), "fd")

# Skewed data — Doane corrects for skewness
calc_bin_width(rexp(200), "doane")

# Discrete / zero-inflated data — FD falls back to Sturges
calc_bin_width(c(rep(0, 50), rep(1, 50)), "fd")

# NA values are silently dropped
calc_bin_width(c(NA, rnorm(100), NA), "scott")

# Compare all methods on the same data
x <- rnorm(500)
methods <- c("auto", "sturges", "fd", "sqrt", "rice", "scott", "doane")
vapply(methods, \(m) calc_bin_width(x, m), numeric(1))

# Error conditions
try(calc_bin_width(letters))               # non-numeric x
try(calc_bin_width(c(1, NA, NA)))          # only one non-NA value
try(calc_bin_width(rep(5, 50)))            # zero range
try(calc_bin_width(rnorm(200), "badrule")) # unknown method
try(calc_bin_width(c(1, 2), "doane"))      # doane needs n >= 3
```

`calc_CH4sat` *Calculate dissolved methane saturation*

Description

Calculates the dissolved methane concentration in equilibrium with the atmosphere from water temperature, barometric pressure, and salinity, using the atmospheric-equilibrium solubility equation of Wiesenburg and Guinasso (1979).

Usage

```
calc_CH4sat(
  temp_water,
  atmo_press,
  units = "atm",
  salinity = 0,
  xCH4_ppm = 1.9,
  out_units = "umol/L"
)
```

Arguments

`temp_water`	Numeric vector. Water temperature in degrees Celsius.
`atmo_press`	Numeric vector. Barometric pressure at the site.
`units`	Character string. Units of `atmo_press`. See `convert_pressure()` for accepted pressure units. Defaults to `"atm"`.
`salinity`	Numeric vector. Salinity in parts per thousand. Defaults to freshwater (`0`).
`xCH4_ppm`	Numeric vector. Dry-air mole fraction of CH4 in parts per million. Defaults to `1.9`, an approximate present-day value; override it with a measured atmospheric value where available.
`out_units`	Character string. Output concentration units, either `"umol/L"` (the default) or `"mg/L"`.

Details

The equilibrium concentration follows Wiesenburg and Guinasso (1979) eq 7 with their volumetric (nmol/L) coefficients (Table VI), in which the dry-air mole fraction enters the solubility equation directly. A vapor-pressure correction scales the 1 atm result to the supplied barometric pressure.

Value

Numeric vector of dissolved CH4 saturation in the requested units.

References

Wiesenburg, D.A., and Guinasso, N.L. (1979). Equilibrium solubilities of methane, carbon monoxide, and hydrogen in water and sea water. Journal of Chemical and Engineering Data, 24(4), 356-360.

See Also

calc_O2sat(), calc_CO2sat(), calc_N2Osat()

Examples

```
calc_CH4sat(temp_water = 20, atmo_press = 1, salinity = 0)

calc_CH4sat(
  temp_water = c(5, 15, 25),
  atmo_press = 101.325,
  units = "kPa",
  xCH4_ppm = 1.9,
  out_units = "mg/L"
)
```

---

calc_CO2_mgL *Calculate dissolved CO2 concentration in mg/L*

Description

Converts a measured or modeled CO2 mole fraction to dissolved CO2 concentration in mg/L. This is a volume-based companion to calc_CO2_molKg() and uses water density to convert from mol/kg.

Usage

```
calc_CO2_mgL(
  CO2_ppm,
  temp_water,
  waterDepth_m,
  atmo_press,
  press_units,
  salinity = 0
)
```

Arguments

CO2_ppm	Numeric vector. Mole fraction of CO2 in air in parts per million.
temp_water	Numeric vector. Water temperature in degrees Celsius.
waterDepth_m	Numeric vector. Water depth above the sensor in meters.
atmo_press	Numeric vector. Atmospheric pressure at the water surface.

press_units	Character string giving the units of atmo_press. See convert_pressure() for accepted pressure units.
salinity	Numeric vector. Salinity in practical salinity units. Defaults to freshwater (0).

Value

Numeric vector of dissolved CO2 concentration in mg/L.

See Also

calc_CO2_molKg(), xCO2_to_pCO2(), calc_water_density()

Examples

```
calc_CO2_mgL(
  CO2_ppm = c(420, 800, 1200),
  temp_water = 20,
  waterDepth_m = 0.5,
  atmo_press = 101.325,
  press_units = "kPa"
)
```

---

calc_CO2_molKg *Calculate dissolved CO2 concentration in mol/kg*

Description

Converts a measured or modeled CO2 mole fraction to dissolved CO2 concentration using atmospheric pressure, water-column pressure, vapor pressure, and the Weiss (1974) CO2 solubility coefficient.

Usage

```
calc_CO2_molKg(
  CO2_ppm,
  temp_water,
  waterDepth_m,
  atmo_press,
  press_units,
  salinity = 0
)
```

Arguments

CO2_ppm	Numeric vector. Mole fraction of CO2 in air in parts per million.
temp_water	Numeric vector. Water temperature in degrees Celsius.
waterDepth_m	Numeric vector. Water depth above the sensor in meters.
atmo_press	Numeric vector. Atmospheric pressure at the water surface.
press_units	Character string giving the units of atmo_press. See convert_pressure() for accepted pressure units.
salinity	Numeric vector. Salinity in practical salinity units. Defaults to freshwater (0).

Value

Numeric vector of dissolved CO2 concentration in mol/kg.

See Also

calc_CO2_mgL(), xCO2_to_pCO2(), calc_K0()

Examples

```
calc_CO2_molKg(
  CO2_ppm = 420,
  temp_water = 20,
  waterDepth_m = 0.5,
  atmo_press = 101.325,
  press_units = "kPa"
)
```

---

calc_CO2sat *Calculate dissolved carbon dioxide saturation*

Description

Calculates the dissolved carbon dioxide concentration in equilibrium with the atmosphere from water temperature, barometric pressure, and salinity, using the Weiss and Price (1980) solubility function for carbon dioxide as a non-ideal atmospheric trace gas.

Usage

```
calc_CO2sat(
  temp_water,
  atmo_press,
  units = "atm",
  salinity = 0,
  xCO2_ppm = 420,
  out_units = "umol/L"
)
```

Arguments

temp_water	Numeric vector. Water temperature in degrees Celsius.
atmo_press	Numeric vector. Barometric pressure at the site.
units	Character string. Units of atmo_press. See convert_pressure() for accepted pressure units. Defaults to "atm".
salinity	Numeric vector. Salinity in parts per thousand. Defaults to freshwater (0).
xCO2_ppm	Numeric vector. Dry-air mole fraction of CO2 in parts per million. Defaults to 420, an approximate present-day value; override it with a measured atmospheric value where available.
out_units	Character string. Output concentration units, either "umol/L" (the default) or "mg/L".

Details

The equilibrium concentration is $C^* = x'F$, where x' is the dry-air mole fraction and F is the moist-air solubility function of Weiss and Price (1980), evaluated with their volumetric coefficients (Table VI). A vapor-pressure correction scales the 1 atm result to the supplied barometric pressure. To convert a *measured* CO2 mole fraction (rather than the atmospheric value) to a dissolved concentration, see calc_CO2_molKg().

Value

Numeric vector of dissolved CO2 saturation in the requested units.

References

Weiss, R.F., and Price, B.A. (1980). Nitrous oxide solubility in water and seawater. Marine Chemistry, 8, 347-359.

See Also

calc_O2sat(), calc_CH4sat(), calc_N2Osat(), calc_CO2_molKg()

Examples

```
calc_CO2sat(temp_water = 20, atmo_press = 1, salinity = 0)

calc_CO2sat(
  temp_water = c(5, 15, 25),
  atmo_press = 101.325,
  units = "kPa",
  xCO2_ppm = 420,
  out_units = "mg/L"
)
```

calc_cv *Calculate the coefficient of variation*

Description

Computes the coefficient of variation (CV), a unitless measure of relative variability. By default, the result is returned as a percentage.

Usage

```
calc_cv(x, na.rm = TRUE, as_percent = TRUE, robust = FALSE)
```

Arguments

x	Numeric vector.
na.rm	Logical. If TRUE, remove missing values before calculation. Defaults to TRUE.
as_percent	Logical. If TRUE, multiply the CV by 100. Defaults to TRUE.
robust	Logical. If TRUE, use median and MAD instead of mean and standard deviation. Defaults to FALSE.

Value

A single numeric CV value, or NA_real_ when the input is non-numeric, empty, all missing after NA removal, or centered on zero. Problematic inputs also produce a warning.

Examples

```
discharge <- c(0.12, 0.18, 0.15, 1.4, 0.09)

calc_cv(discharge)
calc_cv(discharge, as_percent = FALSE)
calc_cv(discharge, robust = TRUE)
```

calc_exceedance_prob *Calculate flow exceedance probabilities*

Description

Calculates exceedance probabilities for a discharge vector with the Weibull plotting-position formula. Higher flows receive lower exceedance probabilities.

Usage

```
calc_exceedance_prob(flow, rm.zero = FALSE)
```

Arguments

`flow`	Numeric vector of flow or discharge values.
`rm.zero`	Logical. If `TRUE`, zero and negative values are omitted from the ranking and returned as `NA`. Defaults to `FALSE`.

Details

The Weibull plotting position is:

$$P = \frac{\text{rank}}{n+1}$$

where `rank` is the descending rank of the flow value and `n` is the number of ranked observations.

Value

Numeric vector of exceedance probabilities with the same length as `flow`. Missing input values return `NA`.

Examples

```
flow_data <- c(10, 5, 0, 15, 8, NA, 0, 20)

calc_exceedance_prob(flow_data)
calc_exceedance_prob(flow_data, rm.zero = TRUE)
```

`calc_K0` *Calculate the CO2 solubility coefficient*

Description

Calculates the Weiss (1974) CO2 solubility coefficient, `K0`, for water at the supplied temperature, salinity, atmospheric pressure, and water depth. The depth and atmospheric pressure terms apply a pressure correction for the total pressure at the sensor.

Usage

```
calc_K0(temp_water, waterDepth_m = 0, atmo_press = 1, salinity = 0)
```

Arguments

`temp_water`	Numeric vector. Water temperature in degrees Celsius.
`waterDepth_m`	Numeric vector. Water depth in meters. Defaults to `0`.
`atmo_press`	Numeric vector. Atmospheric pressure in atm. Defaults to standard atmosphere (`1`).
`salinity`	Numeric vector. Salinity in practical salinity units. Defaults to freshwater (`0`).

Value

Numeric vector of K0 values in mol/(kg atm).

References

Weiss, R.F. (1974). Carbon dioxide in water and seawater: the solubility of a non-ideal gas. Marine Chemistry, 2, 203-215.

Examples

```
calc_K0(temp_water = 20)
calc_K0(temp_water = c(5, 15, 25), waterDepth_m = 1, salinity = 0)
```

---

calc_mode *Calculate the mode of a vector*

Description

Finds the most frequent value in an atomic vector. Ties can be handled by returning the first mode, last mode, a random mode, or all modes.

Usage

```
calc_mode(x, na.rm = TRUE, multi = "first")
```

Arguments

x	Atomic vector, such as numeric, character, logical, or factor.
na.rm	Logical. If TRUE, remove missing values before calculation. Defaults to TRUE.
multi	Character string controlling ties. Options are: "first" Return the first mode in sorted table order. "last" Return the last mode in sorted table order. "sample" Return one randomly selected mode. "all" Return all tied modes.

Details

For factors, returned mode values preserve the original levels. When na.rm = FALSE, missing values are included in the frequency table.

Value

A value or vector with the same general type as x. Returns NA for empty inputs or inputs containing only missing values after NA removal.

See Also

`table()` for frequency tables.

Examples

```
calc_mode(c(1, 2, 2, 3, 3, 3))

tied <- c("riffle", "run", "riffle", "pool", "run")
calc_mode(tied)
calc_mode(tied, multi = "all")

fruit <- factor(c("apple", "banana", "banana", "cherry"))
calc_mode(fruit)
```

---

calc_N2Osat *Calculate dissolved nitrous oxide saturation*

Description

Calculates the dissolved nitrous oxide concentration in equilibrium with the atmosphere from water temperature, barometric pressure, and salinity, using the Weiss and Price (1980) solubility function for nitrous oxide as a non-ideal atmospheric trace gas.

Usage

```
calc_N2Osat(
  temp_water,
  atmo_press,
  units = "atm",
  salinity = 0,
  xN2O_ppm = 0.338,
  out_units = "umol/L"
)
```

Arguments

`temp_water`	Numeric vector. Water temperature in degrees Celsius.
`atmo_press`	Numeric vector. Barometric pressure at the site.
`units`	Character string. Units of `atmo_press`. See `convert_pressure()` for accepted pressure units. Defaults to `"atm"`.
`salinity`	Numeric vector. Salinity in parts per thousand. Defaults to freshwater (`0`).
`xN2O_ppm`	Numeric vector. Dry-air mole fraction of N2O in parts per million. Defaults to `0.338`, an approximate present-day value; override it with a measured atmospheric value where available.
`out_units`	Character string. Output concentration units, either `"umol/L"` (the default) or `"mg/L"`.

Details

The equilibrium concentration is $C^* = x'F$, where x' is the dry-air mole fraction and F is the moist-air solubility function of Weiss and Price (1980), evaluated with their volumetric coefficients (Table II). A vapor-pressure correction scales the 1 atm result to the supplied barometric pressure.

Value

Numeric vector of dissolved N2O saturation in the requested units.

References

Weiss, R.F., and Price, B.A. (1980). Nitrous oxide solubility in water and seawater. Marine Chemistry, 8, 347-359.

See Also

`calc_O2sat()`, `calc_CO2sat()`, `calc_CH4sat()`

Examples

```
calc_N2Osat(temp_water = 20, atmo_press = 1, salinity = 0)

calc_N2Osat(
  temp_water = c(5, 15, 25),
  atmo_press = 101.325,
  units = "kPa",
  xN2O_ppm = 0.338,
  out_units = "mg/L"
)
```

`calc_O2sat` *Calculate dissolved oxygen saturation*

Description

Calculates the dissolved oxygen concentration in equilibrium with the atmosphere from water temperature, barometric pressure, and salinity. This is the `DO.sat` quantity commonly required by stream metabolism models.

Usage

```
calc_O2sat(
  temp_water,
  atmo_press,
  units = "atm",
  salinity = 0,
  out_units = "mg/L"
)
```

Arguments

temp_water	Numeric vector. Water temperature in degrees Celsius.
atmo_press	Numeric vector. Barometric pressure at the site.
units	Character string. Units of atmo_press. See convert_pressure() for accepted pressure units. Defaults to "atm".
salinity	Numeric vector. Salinity in parts per thousand. Defaults to freshwater (0).
out_units	Character string. Output concentration units, either "mg/L" (the default) or "umol/L".

Details

The calculation uses the Benson and Krause umol/kg fit reported by Garcia and Gordon (1992), applies a vapor-pressure correction for non-standard barometric pressure, and converts from mass-based concentration to a volume basis with a salinity-aware density correction.

Value

Numeric vector of dissolved oxygen saturation in the requested units.

References

Garcia, H.E., and Gordon, L.I. (1992). Oxygen solubility in seawater: better fitting equations. Limnology and Oceanography, 37(6), 1307-1312.

See Also

calc_CO2sat(), calc_CH4sat(), calc_N2Osat()

Examples

```
calc_O2sat(temp_water = 15, atmo_press = 1, units = "atm")

calc_O2sat(
  temp_water = c(5, 15, 25),
  atmo_press = c(101.2, 100.8, 100.5),
  units = "kPa"
)

# Return micromoles per liter instead of mg/L
calc_O2sat(temp_water = 15, atmo_press = 1, out_units = "umol/L")
```

`calc_par` *Calculate modeled photosynthetically active radiation*

Description

Calculate photosynthetically active radiation (PAR) for a series of date-times and site coordinates. Input times should be mean solar time, as expected by stream metabolism models. Renamed from `calc_light()` to avoid shadowing `streamMetabolizer::calc_light()`.

Usage

```
calc_par(solar_time, latitude, longitude, max_par = 2326)
```

Arguments

`solar_time`	POSIXct vector of mean solar time values.
`latitude`	Numeric. Site latitude in decimal degrees between -90 and 90.
`longitude`	Numeric. Site longitude in decimal degrees; western longitudes are negative.
`max_par`	Numeric. Peak daily PAR in umol m^-2 s^-1. Defaults to `2326`.

Details

This function was adapted from `streamMetabolizer::calc_light` and is included here under the terms of the CC0 1.0 Universal public domain dedication, as described at: https://www.usgs.gov/information-policies-and-instructions/copyrights-and-credits

Original citation: Appling, A. P., Hall, R. O., Yackulic, C. B., & Arroita, M. (2018). Overcoming Equifinality: Leveraging Long Time Series for Stream Metabolism Estimation. *Journal of Geophysical Research: Biogeosciences*, 123(2), 624-645. https://doi.org/10.1002/2017JG004140

The original implementation routed PAR through SW and back via the `LakeMetabolizer::par.to.sw.base()` / `sw.to.par.base()` linear factors (`par * 0.473` and `sw * 2.114`). Because those factors are reciprocals the round-trip cancels exactly, so the conversion has been inlined and the `streamMetabolizer` / `LakeMetabolizer` dependency removed from this function.

The solar zenith angle at each timestamp is computed via `SunCalcMeeus::sun_zenith_angle()` (Meeus's algorithms), and PAR is modeled as `max_par * cos(zenith)` clipped at zero. The mean solar input is converted back to UTC for the zenith calculation via the standard 15 deg / hour longitude offset.

Value

Numeric vector of modeled PAR in umol m^-2 s^-1.

References

Britton, C. M. and Dodd, J. D. (1976). Relationships of photosynthetically active radiation and shortwave irradiance. *Agricultural Meteorology*, 17(1), 1-7. (Source of the 0.473 W m^-2 per umol m^-2 s^-1 PAR-SW factor.)

Examples

```
utc <- as.POSIXct("2024-06-21 18:00:00", tz = "UTC")
solar_time <- convert_to_solar_time(utc, longitude = -96.6)

calc_par(solar_time, latitude = 39.1, longitude = -96.6)
```

`calc_vapor_press` *Calculate water vapor pressure*

Description

Calculates the vapor pressure of freshwater or seawater in atmospheres.

Usage

```
calc_vapor_press(temp_water = 25, salinity = 0, method = "Dickson2007")
```

Arguments

`temp_water`	Numeric vector. Water temperature in degrees Celsius. Defaults to `25`.
`salinity`	Numeric vector. Salinity in practical salinity units. Defaults to freshwater (`0`).
`method`	Character string. Calculation method. Use `"Dickson2007"` for seawater and `"MIMSY"` for freshwater. Defaults to `"Dickson2007"`.

Details

`"Dickson2007"` follows the ocean CO2 best-practices guide and includes a salinity correction. `"MIMSY"` uses the Antoine equation for freshwater.

Value

Numeric vector of vapor pressure in atm.

References

Dickson, A.G., Sabine, C.L., and Christian, J.R. (Eds.) (2007). Guide to best practices for ocean CO2 measurements. PICES Special Publication 3.

Kelly, M.C. (2024). mimsy: Calculate MIMS Dissolved Gas Concentrations Without Getting a Headache. R package version 0.6.5. https://CRAN.R-project.org/package=mimsy

Examples

```
calc_vapor_press(temp_water = 25, salinity = 35, method = "Dickson2007")
calc_vapor_press(temp_water = 20, salinity = 0, method = "MIMSY")
```

calc_water_density *Calculate water density*

Description

Calculates the density of pure water (kg/m^3) from temperature in degrees Celsius using the polynomial approximation of Kell (1975). Valid for temperatures between 0°C and 150°C at atmospheric pressure.

Usage

```
calc_water_density(water_temp)
```

Arguments

water_temp	Numeric vector. Water temperature in degrees Celsius.

Value

Numeric vector of water density in kg/m^3.

References

Kell, G. S. (1975). Density, thermal expansivity, and compressibility of liquid water from 0° to 150°C: Correlations and tables for atmospheric pressure and saturation reviewed and expressed on 1968 temperature scale. *Journal of Chemical and Engineering Data*, 20(1), 97–105. doi:10.1021/je60064a005

Examples

```
calc_water_density(15)
calc_water_density(c(0, 10, 20, 30))
```

calc_water_height *Calculate water height from pressure*

Description

Converts pressure readings from vented or unvented water-level sensors to water height in meters.

Usage

```
calc_water_height(sensor_kPa, atmo_kPa = NULL, water_temp, type = "vented")
```

Arguments

sensor_kPa	Numeric vector. Sensor pressure in kilopascals. For type = "vented", this is differential pressure from the water column. For type = "unvented", this is absolute pressure.
atmo_kPa	Numeric vector. Atmospheric pressure in kilopascals. Required when type = "unvented".
water_temp	Numeric vector. Water temperature in degrees Celsius.
type	Character string. Sensor type, either "vented" or "unvented". Defaults to "vented".

Details

The pressure difference is divided by water density and gravitational acceleration. Water density is calculated with calc_water_density().

Value

Numeric vector of water height in meters.

References

Kell, G. S. (1975). Density, thermal expansivity, and compressibility of liquid water from 0° to 150°C: Correlations and tables for atmospheric pressure and saturation reviewed and expressed on 1968 temperature scale. *Journal of Chemical and Engineering Data*, 20(1), 97-105. doi:10.1021/je60064a005

Examples

```
calc_water_height(sensor_kPa = 19.2, water_temp = 15, type = "vented")

calc_water_height(
  sensor_kPa = 120.5,
  atmo_kPa = 101.3,
  water_temp = 15,
  type = "unvented"
)
```

---

cdo *Query the NCEI Climate Data Online (CDO) Web Services v2 API*

Description

These functions wrap the seven endpoints of NOAA's Climate Data Online Web Services v2 API (https://www.ncei.noaa.gov/cdo-web/api/v2). They retrieve observations (cdo_data()), search the station catalog (cdo_stations()), and browse the metadata catalog (cdo_datasets(), cdo_datacategories(), cdo_datatypes(), cdo_locationcategories(), cdo_locations()).

Usage

```
cdo_datasets(
  id = NULL,
  datatypeid = NULL,
  locationid = NULL,
  stationid = NULL,
  startdate = NULL,
  enddate = NULL,
  sortfield = NULL,
  sortorder = NULL,
  max_results = Inf
)

cdo_datacategories(
  id = NULL,
  datasetid = NULL,
  locationid = NULL,
  stationid = NULL,
  startdate = NULL,
  enddate = NULL,
  sortfield = NULL,
  sortorder = NULL,
  max_results = Inf
)

cdo_datatypes(
  id = NULL,
  datasetid = NULL,
  locationid = NULL,
  stationid = NULL,
  datacategoryid = NULL,
  startdate = NULL,
  enddate = NULL,
  sortfield = NULL,
  sortorder = NULL,
  max_results = Inf
)

cdo_locationcategories(
  id = NULL,
  datasetid = NULL,
  startdate = NULL,
  enddate = NULL,
  sortfield = NULL,
  sortorder = NULL,
  max_results = Inf
)
```

```
cdo_locations(
  id = NULL,
  datasetid = NULL,
  locationcategoryid = NULL,
  datacategoryid = NULL,
  startdate = NULL,
  enddate = NULL,
  sortfield = NULL,
  sortorder = NULL,
  max_results = Inf
)

cdo_stations(
  id = NULL,
  datasetid = NULL,
  locationid = NULL,
  datacategoryid = NULL,
  datatypeid = NULL,
  extent = NULL,
  startdate = NULL,
  enddate = NULL,
  sortfield = NULL,
  sortorder = NULL,
  max_results = Inf
)

cdo_data(
  datasetid,
  startdate,
  enddate,
  datatypeid = NULL,
  locationid = NULL,
  stationid = NULL,
  units = "metric",
  sortfield = NULL,
  sortorder = NULL,
  includemetadata = FALSE,
  max_results = Inf
)
```

Arguments

id	Optional character string. When supplied, fetches a single resource by its identifier (e.g., `cdo_datasets("GHCND")`) and returns a named list. When `NULL` (default) the list endpoint is queried and a tibble is returned.
startdate, enddate	Optional date filters as `Date` objects or `"YYYY-MM-DD"` strings. For `cdo_data()` both are required.

`sortfield`, `sortorder`
: Optional sort controls. `sortfield` is one of `"id"`, `"name"`, `"mindate"`, `"maxdate"`, `"datacoverage"`; `sortorder` is `"asc"` (default) or `"desc"`.

`max_results`
: Maximum number of rows to return. Defaults to `Inf` (all matching results).

`datasetid`, `datatypeid`, `locationid`, `stationid`, `datacategoryid`, `locationcategoryid`
: Optional character vectors filtering the result set. Multiple values are sent as repeated query parameters.

`extent`
: For `cdo_stations()` only. Numeric vector of length 4 giving a bounding box as `c(min_lat, min_lon, max_lat, max_lon)`.

`units`
: For `cdo_data()` only. `"metric"` (default) or `"standard"`.

`includemetadata`
: For `cdo_data()` only. Logical; when `FALSE` (default) the API skips computing the result-set count, which can noticeably speed up large requests.

Value

A tibble for list queries, or a named list for single-resource (`id`-based) queries. Date-like columns (`mindate`, `maxdate`, `date`) are parsed to `Date` / `POSIXct` and numeric columns (`latitude`, `longitude`, `elevation`, `value`, `datacoverage`) are coerced to numeric.

Authentication

All CDO endpoints require a free API token. Request one at https://www.ncdc.noaa.gov/cdo-web/token and store it in the `API_NCEI_CDO` environment variable, for example by adding `API_NCEI_CDO=...` to your `~/.Renviron`.

Pagination

List endpoints paginate at 1000 results per page. By default each function transparently walks all pages and returns a single tibble. Use `max_results` to cap the total number of rows returned.

Rate limits

Each token is limited to 5 requests per second and 10,000 requests per day. The 5/s cap is enforced client-side via `httr2::req_throttle()` (requests block briefly rather than failing). The package also tracks a session-local counter; once 10,000 successful requests have been made in the current R session the next call aborts with an error directing you to `cdo_reset_request_count()`. The counter is best-effort and does not include requests made by other R sessions or tools.

See Also

`ncei_data()` and `ncei_stations()` for the parallel NCEI Access Data Service (no token required, different endpoint structure).

Examples

```
## Not run:
cdo_datasets()
cdo_datasets("GHCND")

## End(Not run)
## Not run:
cdo_datacategories(datasetid = "GHCND")

## End(Not run)
## Not run:
cdo_datatypes(datasetid = "GHCND", datacategoryid = "TEMP")
cdo_datatypes("TMAX")

## End(Not run)
## Not run:
cdo_locationcategories()
cdo_locationcategories("ST")

## End(Not run)
## Not run:
cdo_locations(locationcategoryid = "ST")
cdo_locations("FIPS:37")

## End(Not run)
## Not run:
cdo_stations(locationid = "FIPS:37", datasetid = "GHCND")
cdo_stations("GHCND:USW00013722")

## End(Not run)
## Not run:
cdo_data(
  datasetid = "GHCND",
  stationid = "GHCND:USW00013722",
  startdate = "2024-01-01",
  enddate   = "2024-01-31",
  datatypeid = c("TMAX", "TMIN")
)

## End(Not run)
```

`cdo_request_count` *Inspect and reset the NCEI CDO session request counter*

Description

The NCEI CDO Web Services v2 API enforces a daily limit of 10,000 requests per token. The package counts every successful CDO request made in the current R session and aborts further requests once the limit is reached. These helpers expose the counter.

Usage

```
cdo_request_count()

cdo_reset_request_count()
```

Details

Because the API does not report remaining quota, the counter is session-local and best-effort: it does not include requests made by other R sessions, other tools, or earlier sessions on the same day.

Value

`cdo_request_count()` returns an integer count of successful CDO requests made this session. `cdo_reset_request_count()` is called for its side effect of zeroing the counter and returns the prior count invisibly.

See Also

`cdo_data()` and friends.

---

closest_noaa_stations *Find NOAA stations near a location*

Description

Identifies NOAA GHCND weather stations within a specified radius of a target location. Station candidates are retrieved via the NCEI Search API using a bounding box and then filtered to the exact circular radius using geodesic distance.

Usage

```
closest_noaa_stations(
  latitude,
  longitude,
  dist_km,
  start_date = NULL,
  end_date = NULL,
  data_types = NULL
)
```

Arguments

`latitude, longitude`	Numeric. Target coordinates in decimal degrees. Latitude must be between -90 and 90; longitude must be between -180 and 180. Western longitudes are negative.
`dist_km`	Numeric. Search radius in kilometres.

`start_date`, `end_date`
: Optional date range. When supplied only stations whose period of record overlaps this range are returned. Accepts `Date` objects or `"YYYY-MM-DD"` strings.

`data_types`
: Optional character vector of GHCND data type codes (e.g., `c("TMAX", "TMIN")`). When supplied only stations carrying all requested types are returned.

Details

Distances are calculated with `geosphere::distGeo()` using the default WGS84 ellipsoid.

The search first queries the NCEI Search API using a square bounding box of side `2 * dist_km` centred on the target point, then trims the result to the circular radius. This requires one API call and avoids downloading the entire station database.

Value

A tibble of NOAA stations within `dist_km`, sorted by ascending distance. The first column is `distance_km`; remaining columns are those returned by `get_noaa_stations()` (`station_id`, `station_name`, `latitude`, `longitude`, `start_date`, `end_date`). Returns `NULL` when no stations are found within the requested radius.

See Also

`get_noaa_stations()`, `ncei_bbox()`, `ncei_stations()`

Examples

```
## Not run:
# Stations within 50 km of Konza Prairie Biological Station
closest_noaa_stations(
  latitude = 39.1068806,
  longitude = -96.6117151,
  dist_km = 50
)

# Restrict to stations with daily temperature data since 2000
closest_noaa_stations(
  latitude = 39.1068806,
  longitude = -96.6117151,
  dist_km = 100,
  data_types = c("TMAX", "TMIN"),
  start_date = "2000-01-01"
)

## End(Not run)
```

convert_flow *Convert stream discharge between units*

Description

Converts stream discharge measurements between cubic feet per second (cfs), cubic meters per second (cms), and liters per second (lps). Always returns a plain numeric vector.

Usage

```
convert_flow(flow, from, to = "cms")
```

Arguments

flow	Numeric. Stream discharge value(s) to be converted.
from	Character. Units of the input discharge. Accepted values are "cfs" (cubic feet per second), "cms" (cubic meters per second), and "lps" (liters per second).
to	Character. Target unit. Same accepted values as from. Defaults to "cms".

Value

Numeric vector of stream discharge in the requested unit.

Examples

```
convert_flow(14.32, from = "cfs", to = "cms")
convert_flow(14.32, from = "cfs", to = "lps")
convert_flow(c(10, 20, 30), from = "cfs", to = "cms")
```

convert_pressure *Convert barometric pressure between units*

Description

Converts barometric pressure from one unit to another using exact conversion factors. Always returns a plain numeric vector.

Usage

```
convert_pressure(pressure, from, to = "atm")
```

Arguments

pressure	Numeric. Barometric pressure value(s) to be converted.
from	Character. Units of the input barometric pressure. Accepted values are "atm", "hPa", "mbar", "kPa", "Pa", "Torr", "mmHg", "psi", and "bar".
to	Character. Target unit. Same accepted values as from. Defaults to "atm".

Value

Numeric vector of barometric pressure in the requested unit.

Examples

```
convert_pressure(101.3, from = "kPa", to = "atm")
convert_pressure(1013.25, from = "hPa", to = "Pa")
```

---

convert_to_solar_time *Convert a datetime to local solar time*

Description

Converts a datetime to mean or apparent (true) solar time at a given longitude. Mean solar time uses a constant 15 deg/hour longitude offset. Apparent solar time additionally applies the equation of time and is computed via SunCalcMeeus::solar_time().

Usage

```
convert_to_solar_time(dateTime, longitude, type = c("mean", "apparent"))

convert_from_solar_time(
  solar_datetime,
  longitude,
  type = c("mean", "apparent")
)
```

Arguments

dateTime	A datetime vector. POSIXct in any time zone is accepted; the function operates on the underlying instant, so a CDT timestamp and the matching UTC timestamp produce identical results. Character or Date input is coerced with as.POSIXct(., tz = "UTC").
longitude	Numeric. Site longitude in decimal degrees; western longitudes are negative.
type	One of "mean" (default) or "apparent".
solar_datetime	A POSIXct vector of solar-time values. The clock reading is interpreted as solar time even though the tzone attribute is UTC; this matches the convention used by streamMetabolizer and SunCalcMeeus.

Details

These helpers are named `convert_to_solar_time()` / `convert_from_solar_time()` to avoid shadowing the original `streamMetabolizer::convert_UTC_to_solartime()` and `streamMetabolizer::convert_solar` which can still be called directly from `streamMetabolizer` if you need their behaviour.

The mean offset is `longitude / 15 * 3600` seconds. The apparent path delegates to `SunCalcMeeus::solar_time()`, which applies the equation of time using Meeus's algorithms.

`convert_from_solar_time()` inverts the forward conversion by computing the forward offset *at the input solar instant* and subtracting it. This is exact for `type = "mean"`. For `type = "apparent"` it is accurate to within roughly the daily change in the equation of time (~30 s/day), which is well below typical sensor sampling intervals.

Value

A POSIXct vector with class `c("solar_date", "POSIXct", "POSIXt")` (`convert_to_solar_time()`) or `c("POSIXct", "POSIXt")` (`convert_from_solar_time()`). The tzone attribute is `"UTC"` in both cases; the clock reading carries the meaning.

See Also

`SunCalcMeeus::solar_time()`, `streamMetabolizer::convert_UTC_to_solartime()`, `streamMetabolizer::convert_`

Examples

```
utc <- as.POSIXct("2024-06-21 18:00:00", tz = "UTC")

convert_to_solar_time(utc, longitude = -96.6)
convert_to_solar_time(utc, longitude = -96.6, type = "apparent")

# A non-UTC POSIXct gives the same result, since the underlying instant
# is what matters.
local <- as.POSIXct("2024-06-21 13:00:00", tz = "America/Chicago")
convert_to_solar_time(local, longitude = -96.6)

solar <- convert_to_solar_time(utc, longitude = -96.6)
convert_from_solar_time(solar, longitude = -96.6)
```

---

correct_bp *Correct barometric pressure for elevation change*

Description

Adjusts barometric pressure from one elevation to another using the barometric formula. Always returns a plain numeric vector.

Usage

```
correct_bp(
  station_bp,
  air_temp,
  station_elev,
  site_elev,
  from_units = "kPa",
  to_units = "kPa"
)
```

Arguments

station_bp	Numeric. Barometric pressure at the station.
air_temp	Numeric. Air temperature at the station in degrees Celsius.
station_elev	Numeric. Elevation of the station in meters.
site_elev	Numeric. Elevation of the target site in meters.
from_units	Character. Units of station_bp. Defaults to "kPa". See convert_pressure() for accepted values.
to_units	Character. Desired output units. Defaults to "kPa". See convert_pressure() for accepted values.

Value

Numeric vector of corrected barometric pressure in to_units.

See Also

convert_pressure() for supported pressure units.

Examples

```
correct_bp(
  station_bp = c(101.3, 100.5), air_temp = c(15, 10),
  station_elev = 300, site_elev = 500
)
```

---

even_timesteps *Fill missing rows in an even time series*

Description

Builds a complete timestamp sequence for logger data and joins the original observations onto it. Missing timestamps become explicit rows with NA values in the measured columns.

Usage

```
even_timesteps(loggerData, datetime_col = "DateTime_UTC", site_col = NULL)
```

Arguments

loggerData	Data frame or tibble containing timestamped logger data.
datetime_col	Character string naming the POSIXct datetime column. Defaults to "DateTime_UTC".
site_col	Optional character string naming a site column. When supplied, each site is completed independently.

Details

The time step is inferred from the sorted unique timestamps for each site. Use this after removing obvious duplicate or invalid timestamps.

Value

A data frame or tibble, matching the input class, with all original rows plus inserted NA rows for missing time steps.

Examples

```
df <- data.frame(
  DateTime_UTC = as.POSIXct(
    c("2024-01-01 00:00", "2024-01-01 01:00", "2024-01-01 03:00"),
    tz = "UTC"
  ),
  temp_C = c(10.1, 10.4, 10.5)
)

even_timesteps(df)

df_multi <- data.frame(
  DateTime_UTC = c(
    as.POSIXct(
      c("2024-01-01 00:00", "2024-01-01 01:00", "2024-01-01 03:00"),
      tz = "UTC"
    ),
    as.POSIXct(c("2024-01-01 00:00", "2024-01-01 00:30"), tz = "UTC")
  ),
  Site = c("A", "A", "A", "B", "B")
)

even_timesteps(df_multi, site_col = "Site")
```

flag_z *Flag outliers with robust Z-scores*

Description

Identifies potential outliers in a numeric vector based on a moving window robust Z-score approach. The robust Z-score is computed using a biweight scale estimate centered on the median.

Usage

```
flag_z(x, width = 5, threshold = 3, return_z = FALSE)
```

Arguments

x	Numeric vector to check for outliers.
width	Odd integer giving the moving-window width. Defaults to 5.
threshold	Numeric threshold for the absolute Z-score above which a value is flagged. Defaults to 3.
return_z	Logical. If TRUE, return both Z-scores and flags. If FALSE, return only flags. Defaults to FALSE.

Details

For each value in x, a window of length width centered on that value is extracted. The function:

- Computes the median of the window.
- Calculates residuals from the median and MAD-based scale.
- Estimates a robust scale using the Tukey biweight midvariance with tuning constant c = 9 (Mosteller & Tukey 1977; Lax 1985):

$$s^2 = n \cdot \frac{\sum r_i^2 (1 - u_i^2)^4}{\left[\sum (1 - u_i^2)(1 - 5u_i^2)\right]^2}$$

  where $u_i = r_i / (c \cdot \mathrm{MAD})$ and the sums run over $|u_i| < 1$.
- Computes a Z-score as $(x_i - \text{median})/s$.

If the absolute value of the Z-score exceeds threshold, the value is flagged with "Z".

NA values are ignored in the window statistics but retained in output positions.

Value

If return_z = TRUE, a list with:

z A numeric vector of robust Z-scores (with NA where not computable).

flag A character vector of the same length as x, with "Z" where an outlier is detected and NA otherwise.

If return_z = FALSE, only the flag vector is returned.

References

Mosteller, F. and Tukey, J. W. (1977). *Data Analysis and Regression*. Addison-Wesley.

Lax, D. A. (1985). Robust estimators of scale: Finite-sample performance in long-tailed symmetric distributions. *Journal of the American Statistical Association*, 80(391), 736–741.

See Also

`stats::mad()`, `stats::median()`

Examples

```
x <- c(1, 2, 1.5, 1.2, 100, 1.1, 1.3, 1.4)
flag_z(x)
flag_z(x, return_z = TRUE)
```

---

french_creek *French Creek stream metabolism data*

Description

Dissolved oxygen and water temperature measured at 5-minute intervals on French Creek near Laramie, Wyoming, USA during summer/fall 2012. Data were collected as part of a stream metabolism study.

Usage

```
french_creek
```

Format

A tibble with 10,883 rows and 4 columns:

datetime POSIXct. Date and time in UTC (original timezone: MDT, UTC-6).

sonde Character. Sensor identifier. `"REZN"` was deployed 8/23/2012 through 9/1/2012 12:50 MDT; `"TOWN"` from 9/1/2012 12:55 MDT through 9/30/2012. Approximately 15% of rows have `NA` in `temp_C` and `DO_mgL`.

temp_C Numeric. Water temperature in degrees Celsius. Some anomalous negative values are present in the raw data and have not been filtered.

DO_mgL Numeric. Dissolved oxygen concentration in mg/L.

Details

Site information (constant across all rows):

- **Latitude**: 41.33 N
- **Longitude**: -106.3 W
- **Stream depth**: 0.16 m
- **Elevation**: approximately 3,115 m (10220 ft)
- **Location**: French Creek, Laramie, Wyoming, USA

The two sonde values indicate a mid-study sensor swap on September 1, 2012. For most analyses, filter to a single sonde or treat the transition as a potential data break.

Source

Hotchkiss, E. R., & Hall, R. O., Jr. (2015). Whole-stream 13C tracer addition reveals distinct fates of newly fixed carbon. *Ecology*, 96, 403–416. doi:10.1890/140631.1

`get_ghcnh` *Get GHCNh hourly observations*

Description

Downloads Global Historical Climatology Network hourly (GHCNh) data for one or more stations and a date range, returning a parsed tibble ready for use in stream metabolism modelling.

Usage

```
get_ghcnh(stations, start_date, end_date, quiet = FALSE)
```

Arguments

`stations`	Character vector of GHCNh station identifiers (e.g., `"USW00023183"`). These match the `station_id` column returned by `get_noaa_stations()` and `closest_noaa_stations()`.
`start_date, end_date`	Start and end of the requested period as `Date` objects or `"YYYY-MM-DD"` strings.
`quiet`	Logical. When `TRUE` progress messages are suppressed. Default `FALSE`.

Details

GHCNh v1.1.0 data are served as pipe-separated (PSV) files from the NOAA NCEI archive, one file per station per year. Data are current through the present year. The function downloads all station-year combinations covering the requested date range in parallel.

Sentinel values of `-9999` are converted to `NA`. Columns that are all `NA` are removed.

Multiple stations and years are downloaded in parallel using up to four concurrent connections.

Value

A tibble with one row per station-hour. Key columns include:

`station_id` Station identifier.

`station_name` Station name (when available).

`datetime` Observation time as a UTC `POSIXct`.

`temperature` Air temperature in degrees Celsius.

`dew_point_temperature` Dew-point temperature (°C).

`relative_humidity` Relative humidity (%).

`wind_direction` Wind direction (degrees clockwise from north).

`wind_speed` Wind speed (m/s).

`sea_level_pressure` Sea-level pressure (hPa).

`precipitation` Precipitation (mm).

Columns that are entirely `NA` across all stations and hours are dropped from the result.

See Also

`get_noaa_stations()`, `closest_noaa_stations()`, `ncei_data()`

Examples

```
## Not run:
# Fetch WY 2025 hourly data for a station near Konza Prairie
konza <- closest_noaa_stations(
  latitude = 39.1068806,
  longitude = -96.6117151,
  dist_km = 50
)
hourly <- get_ghcnh(
  stations = konza$station_id[1],
  start_date = "2024-10-01",
  end_date = "2025-09-30"
)

## End(Not run)
```

---

get_nasa_data *Download NASA POWER hourly data*

Description

Downloads hourly meteorological data from the NASA POWER project for the date range covered by a time-series data frame and interpolates the result to the input timestamps.

Usage

```
get_nasa_data(
  data,
  datetime_col = "dateTime",
  site_col = NULL,
  latitude = NULL,
  longitude = NULL,
  elev_m = NULL,
  params = c("PSC", "ALLSKY_SFC_SW_DWN", "PRECTOTCORR", "T2M"),
  max_attempts = 5,
  quiet = TRUE,
  lat = lifecycle::deprecated(),
  lon = lifecycle::deprecated()
)
```

Arguments

`data`	A data frame or tibble containing time-series data.
`datetime_col`	Character string specifying the date-time column in `data`. Defaults to `"dateTime"`.
`site_col`	Optional character string specifying the site column in `data`. If `NULL`, `data` is treated as a single site.
`latitude`, `longitude`	Single numeric values used for single-site data. If omitted, `data` may instead contain single-valued `latitude` and `longitude` columns. When `data` contains multiple sites, coordinates must be supplied as per-site columns in `data`.
`elev_m`	Single numeric site elevation in meters. If omitted, `data` may instead contain a single-valued `elev_m` column. When `data` contains multiple sites, elevation must be supplied as a per-site column in `data`.
`params`	Case-insensitive character vector of solar, meteorological, or climatology parameters to download. Defaults to `"PSC"`, `"ALLSKY_SFC_SW_DWN"`, `"PRECTOTCORR"`, and `"T2M"`. See get_power for more information.
`max_attempts`	Number of retry attempts for failed API calls. Default is 5.
`quiet`	Logical. If `TRUE` (default), suppresses progress messages.
`lat`, `lon`	**[Deprecated]** Use `latitude` and `longitude` instead. Data columns named `lat` and `lon` are also deprecated; use `latitude` and `longitude` columns instead.

Value

A tibble interpolated to the non-missing timestamps in `data`, with columns:

Site column Site names in the column named by `site_col`, when supplied

dateTime timestamps from `data` in UTC

PSC Elevation-corrected barometric pressure (kPa)

ALLSKY_SFC_SW_DWN All Sky Surface Shortwave Downward Irradiance (W/m²), when requested

light.obs Observed photosynthetically active radiation (µmol/m²/s), converted from `ALLSKY_SFC_SW_DWN` when requested

T2M Average air temperature at 2 m above the surface (°C)

PRECTOTCORR MERRA-2 bias corrected total precipitation (mm/hr)

Examples

```
## Not run:
stream_data <- data.frame(
  dateTime = as.POSIXct("2024-06-01 12:00:00", tz = "UTC")
)

get_nasa_data(
  stream_data,
  latitude = 39.1,
  longitude = -96.6,
  elev_m = 320
)

site_data <- data.frame(
  site = c("a", "b"),
  dateTime = as.POSIXct(c(
    "2024-06-01 12:00:00",
    "2024-06-01 12:00:00"
  ), tz = "UTC"),
  latitude = c(39.1, 40.0),
  longitude = c(-96.6, -97.2),
  elev_m = c(320, 340)
)

get_nasa_data(site_data, site_col = "site")

## End(Not run)
```

---

get_noaa_stations *Get NOAA station information*

Description

Searches for NOAA weather stations in the NCEI daily-summaries (GHCND) dataset using the NCEI Search Service API. This is the preferred way to find station identifiers before downloading data with `ncei_data()` or `get_ghcnh()`.

Usage

```
get_noaa_stations(
  bbox = NULL,
  start_date = NULL,
```

```
  end_date = NULL,
  data_types = NULL,
  text = NULL,
  limit = 1000L,
  offset = 0L
)
```

Arguments

bbox	Optional numeric vector `c(north, west, south, east)` in decimal degrees. When supplied only stations within this bounding box are returned. Use `ncei_bbox()` to build a bounding box from a centre point and radius. When `NULL` no geographic filter is applied.
start_date, end_date	Optional date range filter. Accepts `Date` objects or `"YYYY-MM-DD"` strings. When supplied only stations whose period of record overlaps this range are returned.
data_types	Optional character vector of GHCND data type codes (e.g., `c("TMAX", "TMIN")`). When supplied only stations carrying all requested types are returned.
text	Optional character string. Filters stations by name.
limit	Integer. Maximum number of stations to return (1–1000, default 1000).
offset	Integer. Zero-based pagination offset (default 0).

Details

Station identifiers in `station_id` are bare GHCND IDs (e.g., `"USW00023183"`) stripped of their `"GHCND:"` prefix. Pass them directly to `ncei_data()` or `get_ghcnh()`.

This function replaces the previous MSHR flat-file approach. Station metadata is now retrieved on demand from the NCEI Search Service API rather than downloaded as a large fixed-width archive.

For distance-based searches use `closest_noaa_stations()`, which builds the bounding box from a latitude, longitude, and search radius.

Value

A tibble with columns `station_id`, `station_name`, `latitude`, `longitude`, `start_date`, and `end_date`. Returns an empty tibble when no stations match the query.

See Also

`closest_noaa_stations()`, `ncei_bbox()`, `ncei_stations()`, `ncei_data()`, `get_ghcnh()`

Examples

```
## Not run:
# All GHCND stations in a region
get_noaa_stations(bbox = c(40, -100, 38, -98))

# Stations with at least temperature data since 2000
get_noaa_stations(
```

```
  data_types = c("TMAX", "TMIN"),
  start_date = "2000-01-01"
)

## End(Not run)
```

`get_season` *Determine the astronomical season from a date*

Description

Classifies dates into astronomical seasons using the precise dates of equinoxes and solstices for each year, computed from Meeus's "Astronomical Algorithms" (Willmann-Bell, 1991), chapter 26.

Usage

```
get_season(date, hemisphere = c("north", "south"), labels = NULL)
```

Arguments

`date`	A `Date`, `POSIXct`, or `POSIXlt` vector, or anything coercible to `Date` via `base::as.Date()`.
`hemisphere`	One of `"north"` (default) or `"south"`.
`labels`	Optional named character vector overriding the default season names. Allowed names are `spring`, `summer`, `autumn`, and `winter`. Partial overrides are allowed; unspecified names keep their defaults.

Details

A mean Julian Ephemeris Day for each equinox and solstice is computed from the Table 26.B polynomial and refined by the 24-term periodic correction in Table 26.C. The corrected instant is accurate to roughly a minute and is then rounded to a calendar date in UTC. The formulas are valid for years 1000-3000.

Southern-hemisphere seasons are the northern seasons offset by six months. For example, dates after the March equinox and before the June solstice are northern spring but southern autumn.

Value

An ordered factor with levels Spring < Summer < Autumn < Winter (or the user-supplied equivalents), the same length as `date`. `NA` in input produces `NA` in output.

Examples

```
get_season(as.Date(c("2024-03-19", "2024-06-20", "2024-09-22", "2024-12-21")))

get_season(Sys.time())

# Southern hemisphere
get_season(as.Date("2024-07-15"), hemisphere = "south")

# Custom labels
get_season(as.Date("2024-09-22"), labels = c(autumn = "Fall"))
```

`get_usgs_elev` *Get elevation from the USGS Elevation Point Query Service*

Description

Queries the USGS Elevation Point Query Service for one or more latitude and longitude pairs. Coordinates are interpreted as WGS84 (WKID 4326). Requests are performed in parallel for improved performance when querying multiple points.

Usage

```
get_usgs_elev(latitude, longitude, units = c("meters", "feet", "m", "ft"))
```

Arguments

`latitude`, `longitude`
: Numeric vectors of latitude and longitude in decimal degrees. Latitude values must be between -90 and 90; longitude values must be between -180 and 180. Western longitudes are negative. Inputs must be the same length.

`units`
: Character scalar specifying the output elevation units. Accepted values are `"meters"`, `"m"`, `"feet"`, and `"ft"`. Matching is case insensitive. Defaults to `"meters"`.

Value

A numeric vector of elevations in the requested units, with one element for each input coordinate pair. If an elevation cannot be retrieved for a point, `NA_real_` is returned for that location and a warning is issued.

Examples

```
## Not run:
get_usgs_elev(
  latitude = 39.102075,
  longitude = -96.594689
)
```

```
get_usgs_elev(
  latitude = c(39.102075, 38.8977),
  longitude = c(-96.594689, -77.0365),
  units = "ft"
)

## End(Not run)
```

`iem_current` *Get current Iowa Environmental Mesonet observations*

Description

Retrieves current observations from the Iowa Environmental Mesonet (IEM).

Usage

```
iem_current(
  network = NULL,
  networkclass = NULL,
  wfo = NULL,
  country = NULL,
  state = NULL,
  stations = NULL,
  minutes = NULL
)
```

Arguments

`network`	Optional single IEM network identifier, such as `"IA_ASOS"`.
`networkclass`	Optional single network class, such as `"ASOS"` or `"COOP"`.
`wfo`	Optional single National Weather Service forecast office identifier, such as `"DMX"`.
`country`	Optional single country identifier, such as `"US"`.
`state`	Optional single state identifier, such as `"IA"`.
`stations`	Optional character vector of station identifiers.
`minutes`	Optional whole number giving the maximum observation age in minutes.

Details

[Experimental]

At least one filter must be supplied. Station identifiers may be shared across networks, so a station-only request can return multiple rows for a single identifier. Use `network` with `stations` when you need one network.

Value

A tibble containing current station metadata and observations returned by IEM, with station identifiers in `station_id`. Units and available variables vary by network.

References

Iowa Environmental Mesonet API: https://mesonet.agron.iastate.edu/api/

IEM API v1 documentation: https://mesonet.agron.iastate.edu/api/1/docs

Examples

```
## Not run:
current <- iem_current(network = "IA_ASOS")
dsm <- iem_current(network = "IA_ASOS", stations = "DSM")

## End(Not run)
```

---

iem_daily *Get Iowa Environmental Mesonet daily summaries*

Description

Retrieves daily summary observations from the Iowa Environmental Mesonet (IEM).

Usage

```
iem_daily(network, station = NULL, date = NULL, year = NULL, month = NULL)
```

Arguments

`network`	A single IEM network identifier, such as `"IA_ASOS"`.
`station`	Optional single station identifier, such as `"DSM"`.
`date`	Optional single local calendar date as a `Date` object or `YYYY-MM-DD` string.
`year`	Optional whole number year.
`month`	Optional whole number month from 1 to 12. Requires `year`.

Details

[Experimental]

Use either `date` for all stations in a network on one local date, or `year` and optional `month` for a longer station or network request. Do not supply `date` together with `year` or `month`.

Value

A tibble containing daily summary observations returned by IEM, with station identifiers in `station_id` when returned. Units and available variables vary by network.

References

Iowa Environmental Mesonet API: https://mesonet.agron.iastate.edu/api/

IEM API v1 documentation: https://mesonet.agron.iastate.edu/api/1/docs

Examples

```
## Not run:
daily <- iem_daily("IA_ASOS", station = "DSM", year = 2024)
network_day <- iem_daily("IA_ASOS", date = "2024-06-01")

## End(Not run)
```

`iem_networks` *Get Iowa Environmental Mesonet network identifiers*

Description

Retrieves the network table from the Iowa Environmental Mesonet (IEM) API.

Usage

```
iem_networks()
```

Details

[Experimental]

The Iowa Environmental Mesonet groups stations into networks such as `IA_ASOS`, `IA_COOP`, and other state, roadway, hydrological, and special observing networks. Use `iem_networks()` to discover valid `network` values for other IEM helpers.

Value

A tibble containing IEM network identifiers in `network`, network names in `network_name`, time zones, geographic extents, and windrose update timestamps when available.

References

Iowa Environmental Mesonet API: https://mesonet.agron.iastate.edu/api/

IEM API v1 documentation: https://mesonet.agron.iastate.edu/api/1/docs

Examples

```
## Not run:
networks <- iem_networks()
subset(networks, id == "IA_ASOS")

## End(Not run)
```

`iem_obhistory` *Get one day of Iowa Environmental Mesonet observations*

Description

Retrieves one local calendar day of observations for one IEM station.

Usage

```
iem_obhistory(station, network, date = Sys.Date(), full = FALSE)
```

Arguments

`station`	A single station identifier, such as `"DSM"`.
`network`	A single IEM network identifier, such as `"IA_ASOS"`.
`date`	A single local calendar date as a `Date` object or `YYYY-MM-DD` string. Defaults to `Sys.Date()`.
`full`	Logical. If `TRUE`, request all available observation fields.

Details

[Experimental]

The `date` argument is interpreted by IEM as a local station calendar date. UTC timestamps are parsed as `POSIXct` values in the UTC time zone. Local timestamp fields are returned as character values because station time zones vary by network.

Value

A tibble containing one day of station observations returned by IEM, with station identifiers in `station_id` when returned. Units and available variables vary by network.

References

Iowa Environmental Mesonet API: https://mesonet.agron.iastate.edu/api/

IEM API v1 documentation: https://mesonet.agron.iastate.edu/api/1/docs

Examples

```
## Not run:
obs <- iem_obhistory("DSM", network = "IA_ASOS", date = "2024-06-01")

## End(Not run)
```

`iem_stations` *Get Iowa Environmental Mesonet station metadata*

Description

Retrieves station metadata for one IEM network or one station identifier.

Usage

```
iem_stations(network)

iem_station(station_id)
```

Arguments

`network`	A single IEM network identifier, such as `"IA_ASOS"`.
`station_id`	A single station identifier, such as `"DSM"`.

Details

[Experimental]

`iem_stations()` returns all stations in one network. `iem_station()` looks up one station identifier across IEM networks, which is useful because some identifiers are shared by more than one network.

Value

A tibble containing station metadata returned by IEM, including station identifiers in `station_id`, names in `station_name`, network identifiers, time zones, archive dates, and longitude and latitude when available.

References

Iowa Environmental Mesonet API: https://mesonet.agron.iastate.edu/api/

IEM API v1 documentation: https://mesonet.agron.iastate.edu/api/1/docs

Examples

```
## Not run:
stations <- iem_stations("IA_ASOS")
iem_station("DSM")

## End(Not run)
```

kings_discharge *Kings Creek daily water data*

Description

Daily USGS water data for Kings Creek at monitoring location USGS-06879650.

Usage

```
kings_discharge
```

Format

A tibble with 716 rows and 5 variables:

monitoring_location_id USGS monitoring location ID.

time Date of observation.

value Observed value. Units depend on parameter_code.

parameter_code USGS parameter code: 00060 = discharge, 00065 = gage height, 00010 = water temperature.

qualifier USGS data qualifier.

Details

The dataset includes daily observations for water year 2025, from 2024-10-01 through 2025-09-30.

Source

USGS Water Data API, retrieved with dataRetrieval::read_waterdata_daily().

ks_meso_fw13 *Get Kansas Mesonet FW13 data*

Description

Retrieves fire weather data in FW13 format for one station and date range.

Usage

```
ks_meso_fw13(station, start_date, end_date)
```

Arguments

station	Station name as a character string.
start_date	Start date for the data retrieval in YYYY-MM-DD or YYYYMMDD format.
end_date	End date for the data retrieval in YYYY-MM-DD or YYYYMMDD format.

Details

[Experimental]

Kansas Mesonet data are preliminary and subject to revision. Cite the Kansas Mesonet when sharing, publishing, or otherwise disseminating data accessed with this function. A suggested citation format is: Kansas Mesonet, year: webpage title. Accessed date, webpage URL. Review the Kansas Mesonet data usage policy before automated use; automated page scraping or data ingesting without written consent is not permitted.

Value

A character vector containing FW13 records.

References

Kansas Mesonet data usage policy: https://mesonet.k-state.edu/about/usage/

Examples

```
## Not run:
ks_meso_fw13(
  station = "Konza Prairie",
  start_date = "2024-04-01",
  end_date = "2024-04-07"
)

## End(Not run)
```

---

ks_meso_most_recent *Get most recent Kansas Mesonet data timestamp*

Description

Retrieves the timestamp of the most recently ingested data for each station for a given observation interval. The returned data includes the `"--all--"` row reported by Kansas Mesonet.

Usage

```
ks_meso_most_recent(interval)
```

Arguments

`interval`	Data interval. Must be one of `"hour"`, `"5min"`, or `"day"`.

Details

[Experimental]

Kansas Mesonet data are preliminary and subject to revision. Cite the Kansas Mesonet when sharing, publishing, or otherwise disseminating data accessed with this function. A suggested citation format is: Kansas Mesonet, year: webpage title. Accessed date, webpage URL. Review the Kansas Mesonet data usage policy before automated use; automated page scraping or data ingesting without written consent is not permitted.

Value

A tibble with station names in `station_name` and most recent observation times in `timestamp`.

References

Kansas Mesonet data usage policy: https://mesonet.k-state.edu/about/usage/

Examples

```
## Not run:
ks_meso_most_recent(interval = "hour")

## End(Not run)
```

`ks_meso_station_activity`

Get Kansas Mesonet station activity

Description

Retrieves activity data for Kansas Mesonet stations, including observation intervals and data spans.

Usage

```
ks_meso_station_activity()
```

Details

[Experimental]

Kansas Mesonet data are preliminary and subject to revision. Cite the Kansas Mesonet when sharing, publishing, or otherwise disseminating data accessed with this function. A suggested citation format is: Kansas Mesonet, year: webpage title. Accessed date, webpage URL. Review the Kansas Mesonet data usage policy before automated use; automated page scraping or data ingesting without written consent is not permitted.

Value

A tibble with station activity details, including station names in `station_name` and start and end observation times.

References

Kansas Mesonet data usage policy: https://mesonet.k-state.edu/about/usage/

Examples

```
## Not run:
activity <- ks_meso_station_activity()
subset(activity, station_name == "Konza Prairie")

## End(Not run)
```

---

ks_meso_stations *Get Kansas Mesonet station information*

Description

Fetches metadata about Kansas Mesonet stations, including location and network details.

Usage

```
ks_meso_stations()
```

Details

[Experimental]

Kansas Mesonet data are preliminary and subject to revision. Cite the Kansas Mesonet when sharing, publishing, or otherwise disseminating data accessed with this function. A suggested citation format is: Kansas Mesonet, year: webpage title. Accessed date, webpage URL. Review the Kansas Mesonet data usage policy before automated use; automated page scraping or data ingesting without written consent is not permitted.

Value

A tibble containing station metadata, including `station_name`, `station_id`, `network`, and `network_name`.

References

Kansas Mesonet data usage policy: https://mesonet.k-state.edu/about/usage/

Examples

```
## Not run:
stations <- ks_meso_stations()
subset(stations, grepl("Konza", station_name))

## End(Not run)
```

`ks_meso_timeseries` *Get Kansas Mesonet time-series data*

Description

Retrieves weather observations for one or more Kansas Mesonet stations or a supported Kansas Mesonet subnetwork.

Usage

```
ks_meso_timeseries(
  stations = NULL,
  network = NULL,
  start_date,
  end_date,
  interval,
  vars = NULL
)
```

Arguments

`stations`	Character vector of station names to retrieve data for. Use `"all"` to retrieve data for all stations. Must be `NULL` when `network` is supplied.
`network`	Network name to retrieve data for, one of: • `"BBW"`: Big Bend Groundwater Management District • `"EBW"`: Equus Beds Groundwater Management District • `"KSRE"`: K-State Research and Extension This is an alternative to `stations`.
`start_date`	Start date for the data retrieval in `YYYY-MM-DD` or `YYYYMMDD` format.
`end_date`	End date for the data retrieval in `YYYY-MM-DD` or `YYYYMMDD` format.
`interval`	Data interval. Must be one of `"hour"`, `"5min"`, or `"day"`.
`vars`	Character vector of variables to retrieve. Defaults to common weather variables.

Details

[Experimental]

Large date ranges are split into chunks automatically to stay within the Kansas Mesonet API record limit. Data are returned directly and are not written to a local cache.

Kansas Mesonet data are preliminary and subject to revision. Cite the Kansas Mesonet when sharing, publishing, or otherwise disseminating data accessed with this function. A suggested citation format is: Kansas Mesonet, year: webpage title. Accessed date, webpage URL. Review the Kansas Mesonet data usage policy before automated use; automated page scraping or data ingesting without written consent is not permitted.

Value

A tibble containing requested observations with snake_case column names. Kansas Mesonet timestamps are parsed as `POSIXct` values in the fixed Mesonet time zone, `Etc/GMT+6`.

References

Kansas Mesonet data usage policy: https://mesonet.k-state.edu/about/usage/

Examples

```
## Not run:
konza <- ks_meso_timeseries(
  stations = "Konza Prairie",
  start_date = "2024-06-01",
  end_date = "2024-06-07",
  interval = "hour",
  vars = c("TEMP2MAVG", "RELHUM2MAVG", "PRESSUREAVG")
)

## End(Not run)
```

---

ks_meso_vars *Fetch Kansas Mesonet variable metadata*

Description

Retrieves variable metadata from the Kansas Mesonet REST variables page and returns a tidy table of available variables, including their CSV headings, original variable names, cleaned variable names, units, and descriptions.

Usage

```
ks_meso_vars()
```

Details

[Experimental]

The function downloads the Kansas Mesonet variables page, extracts the embedded JavaScript metadata array, parses variable fields, and standardizes variable names into `tidy_name`. Kansas Mesonet data are preliminary and subject to revision. Cite the Kansas Mesonet when sharing, publishing, or otherwise disseminating data accessed with this function. A suggested citation format is: Kansas Mesonet, year: webpage title. Accessed date, webpage URL. Review the Kansas Mesonet data usage policy before automated use; automated page scraping or data ingesting without written consent is not permitted.

Value

A tibble with one row per Kansas Mesonet variable and the columns:

csv_heading The variable heading used in Mesonet CSV output.

var The original variable name from the Mesonet metadata.

tidy_name A cleaned, snake_case version of `var`.

units The measurement units for the variable.

desc A description of the variable.

References

Kansas Mesonet data usage policy: https://mesonet.k-state.edu/about/usage/

Examples

```
## Not run:
vars <- ks_meso_vars()
vars

## End(Not run)
```

`ncei_bbox` *Compute a bounding box around a point*

Description

Returns a bounding box vector suitable for the bbox argument of `ncei_stations()` and `get_noaa_stations()`.

Usage

```
ncei_bbox(latitude, longitude, dist_km)
```

Arguments

latitude, longitude
: Numeric. Target coordinates in decimal degrees. Latitude must be between -90 and 90; longitude between -180 and 180.

dist_km
: Positive numeric. Half-width of the bounding box in kilometres.

Value

A numeric vector `c(north, west, south, east)`.

Examples

```
# Bounding box 50 km around Konza Prairie
ncei_bbox(39.1, -96.6, 50)
```

---

ncei_data *Get data from the NCEI Data Service API*

Description

Retrieves weather and climate data from NOAA's National Centers for Environmental Information (NCEI) using the Access Data Service API.

Usage

```
ncei_data(
  dataset,
  stations,
  start_date,
  end_date,
  data_types = NULL,
  units = "metric",
  include_station_name = TRUE,
  include_station_location = FALSE
)
```

Arguments

dataset
: Character string. The dataset to query. Common values include `"daily-summaries"` (GHCND), `"global-hourly"` (ISD), `"global-summary-of-the-month"`, and `"global-summary-of-the-year"`. New datasets are added periodically; use `ncei_datasets()` to list what is currently available.

stations
: Character vector of station identifiers, such as those returned by `ncei_stations()` or `get_noaa_stations()`.

start_date, end_date
: Start and end of the requested period, as `Date` objects or `"YYYY-MM-DD"` strings.

`data_types` Optional character vector of data type codes to include (e.g., `c("TMAX", "TMIN", "PRCP")` for daily summaries). When `NULL` all available types are returned.

`units` Character string. `"metric"` (default) or `"standard"`.

`include_station_name`
Logical. When `TRUE` (default), a `station_name` column is included in the result.

`include_station_location`
Logical. When `TRUE`, `latitude`, `longitude`, and `elevation` columns are added. Default `FALSE`.

Details

This function calls `https://www.ncei.noaa.gov/access/services/data/v1`. Data are returned in CSV format and parsed into a tibble.

For `dataset = "global-hourly"` (Integrated Surface Database), the mandatory packed fields are expanded into typed numeric columns with SI units and missing-value sentinels converted to `NA`:

`WND` `wind_direction` (degrees), `wind_direction_quality`, `wind_type_code`, `wind_speed` (m/s), `wind_speed_quality`.

`CIG` `ceiling_height` (m), `ceiling_quality`, `ceiling_determination_code`, `ceiling_cavok`.

`VIS` `visibility` (m), `visibility_quality`, `visibility_variability_code`, `visibility_variability_quality`.

`TMP` `temperature` (°C), `temperature_quality`.

`DEW` `dew_point_temperature` (°C), `dew_point_quality`.

`SLP` `sea_level_pressure` (hPa), `sea_level_pressure_quality`.

`AA1`–`AA4` `precipitation_period_hours` (hr), `precipitation` (mm), `precipitation_condition_code`, `precipitation_quality`.

Value

A tibble with one row per observation and snake_case column names. Leading columns (when present) are `station_id`, `station_name`, `latitude`, `longitude`, `elevation`, and either `datetime` (`POSIXct` UTC, for sub-daily datasets) or `date` (`Date`, for daily and coarser datasets). Remaining columns depend on the requested `dataset` and `data_types`. Columns that are entirely `NA` are dropped.

See Also

`ncei_stations()` to search for station identifiers, `ncei_datasets()` to list available datasets.

Examples

```
## Not run:
# Daily temperature and precipitation at a single station
ncei_data(
  dataset = "daily-summaries",
  stations = "USW00023183",
  start_date = "2023-01-01",
  end_date = "2023-12-31",
```

```
  data_types = c("TMAX", "TMIN", "PRCP")
)

# Hourly ISD observations with expanded mandatory fields
ncei_data(
  dataset = "global-hourly",
  stations = "72469023183",
  start_date = "2023-06-01",
  end_date = "2023-06-30"
)

## End(Not run)
```

`ncei_datasets` *List available NCEI datasets*

Description

Retrieves metadata about a dataset from the NCEI Support Service API, including available data types, spatial coverage, and temporal range.

Usage

```
ncei_datasets(dataset)
```

Arguments

`dataset`	Character string. The dataset identifier, such as `"daily-summaries"` or `"global-hourly"`.

Value

A list of dataset metadata as returned by the Support Service API. The structure varies by dataset but typically includes fields such as `id`, `name`, `dataTypes`, and `extent`.

Examples

```
## Not run:
ncei_datasets("daily-summaries")
ncei_datasets("global-hourly")

## End(Not run)
```

`ncei_stations` *Search for NCEI weather stations*

Description

Searches for stations in NOAA's National Centers for Environmental Information (NCEI) using the Common Access Search Service API.

Usage

```
ncei_stations(
  dataset,
  bbox = NULL,
  start_date = NULL,
  end_date = NULL,
  data_types = NULL,
  text = NULL,
  limit = 100L,
  offset = 0L
)
```

Arguments

`dataset`	Character string. The dataset to search within, such as `"daily-summaries"` (GHCND) or `"global-hourly"` (ISD). See `ncei_datasets()` for more dataset identifiers.
`bbox`	Optional numeric vector of length 4 specifying the geographic search area as `c(north, west, south, east)` in decimal degrees. North and south must be between -90 and 90; west and east between -180 and 180. When `NULL` no geographic filter is applied.
`start_date, end_date`	Optional date range filter. Only stations with data overlapping this period are returned. Accepts `Date` objects or `"YYYY-MM-DD"` strings.
`data_types`	Optional character vector of data type codes. Only stations that include all requested types are returned.
`text`	Optional character string. Filters results to stations whose names contain this text.
`limit`	Integer. Maximum number of stations to return per page (default 100, max 1000).
`offset`	Integer. Zero-based pagination offset (default 0).

Details

This function calls `https://www.ncei.noaa.gov/access/services/search/v1/data`.

To find stations near a specific point, compute a bounding box with `ncei_bbox()` and pass it to `bbox`. For site-level station searches prefer `closest_noaa_stations()`, which computes the bounding box automatically and filters by geodesic distance.

Value

A tibble with one row per station and the following columns:

`station_id` Station identifier (dataset prefix stripped, e.g., `"USW00023183"` rather than `"GHCND:USW00023183"`).

`station_name` Station name.

`latitude`,`longitude` Decimal-degree coordinates.

`start_date`,`end_date` Period of record as `Date` objects.

See Also

`closest_noaa_stations()`, `get_noaa_stations()`, `ncei_bbox()`, `ncei_data()`

Examples

```
## Not run:
# All daily-summaries stations in a region
ncei_stations(
  dataset = "daily-summaries",
  bbox = c(40, -100, 38, -98)
)

# Hourly stations with temperature data and a long record
ncei_stations(
  dataset = "global-hourly",
  start_date = "1990-01-01",
  end_date = Sys.Date()
)

## End(Not run)
```

---

pCO2_to_xCO2 *Convert CO2 partial pressure to mole fraction*

Description

Converts water-vapor-corrected CO2 partial pressure, `pCO2`, to CO2 mole fraction, `xCO2`.

Usage

```
pCO2_to_xCO2(temp_water, pCO2_uatm, atmo_press, press_units, ...)
```

Arguments

temp_water	Numeric vector. Water temperature in degrees Celsius.
pCO2_uatm	Numeric vector. Partial pressure of CO2 in microatmospheres.
atmo_press	Numeric vector. Atmospheric pressure.
press_units	Character string giving the units of atmo_press. See convert_pressure() for accepted pressure units.
...	Additional arguments passed to calc_vapor_press(), such as salinity or method.

Value

Numeric vector of CO2 mole fraction in parts per million.

References

Dickson, A.G., Sabine, C.L., and Christian, J.R. (Eds.) (2007). Guide to best practices for ocean CO2 measurements. PICES Special Publication 3.

Examples

```
pCO2_to_xCO2(
  temp_water = 20,
  pCO2_uatm = 400,
  atmo_press = 101.325,
  press_units = "kPa",
  salinity = 0,
  method = "MIMSY"
)
```

---

rcpp_calc_exceedance_prob

Calculate flow exceedance probabilities with C++

Description

Calculates exceedance probabilities using the same Weibull plotting position method and return shape as calc_exceedance_prob(), but delegates ranking to a C++ implementation.

Usage

```
rcpp_calc_exceedance_prob(flow, rm.zero = FALSE)
```

Arguments

flow	Numeric vector of flow or discharge values.
rm.zero	Logical. If TRUE, zero and negative values are omitted from the ranking and returned as NA. Defaults to FALSE.

Value

A numeric vector of exceedance probabilities. If `rm.zero = TRUE`, the returned vector has the same length as the input with `NA` at positions of zero or negative values.

See Also

`calc_exceedance_prob()`

Examples

```
rcpp_calc_exceedance_prob(c(10, 5, 0, 15, 8, NA, 0, 20))
```

---

read_shp *Read a shapefile or zipped shapefile*

Description

Reads a shapefile or shapefile contained in a zip archive, automatically handling path specifications and layer detection. This is a convenience wrapper around `sf::st_read()` that simplifies reading common shapefile formats.

Usage

```
read_shp(path, layer = NULL)
```

Arguments

`path`	Path to the shapefile (.shp), directory containing shapefile components, or zip file containing a shapefile. Tilde expansion is performed.
`layer`	Layer name to read (optional). If not specified and path points to a .shp file, the layer name will be derived from the filename. For directories or zip files with multiple layers, the first layer will be used with a warning.

Details

The function handles several input types:

- Direct paths to .shp files (automatically extracts directory and layer name)
- Directories containing shapefile components (auto-detects layer)
- Zip files containing shapefiles (uses GDAL's virtual file system)

When reading from zip files, the function uses GDAL's `/vsizip/` virtual file system prefix.

Value

An sf object containing the spatial data from the shapefile.

See Also

sf::st_read() for the underlying reading function, sf::st_layers() for layer detection

Examples

```
## Not run:
# Read from .shp file path
shp_data <- read_shp("path/to/file.shp")

# Read from directory containing shapefile components
shp_data <- read_shp("path/to/shapefile_directory")

# Read from zip file
shp_data <- read_shp("path/to/archive.zip")

# Specify layer explicitly
shp_data <- read_shp("path/to/multi_layer.zip", layer = "specific_layer")

## End(Not run)
```

---

tex_meso_current *Get current TexMesonet data*

Description

Retrieves the most recent observation from each TWDB TexMesonet station.

Usage

```
tex_meso_current()
```

Details

[Experimental]

Current data are near-real-time observations. TexMesonet reports values in the units returned by the API, available with `attr(x, "units")`.

Value

A tibble containing the most recent station observations, including `station_id`, `station_name`, and UTC `recorded_time`. The API returns availability by station, so some measurement columns may contain missing values. The returned tibble has a `"units"` attribute containing units reported by the API.

References

TexMesonet APIs: https://www.texmesonet.org/Apis

Examples

```
## Not run:
current <- tex_meso_current()
attr(current, "units")

## End(Not run)
```

`tex_meso_stations` *Get TexMesonet station information*

Description

Retrieves name, location, and status metadata for Texas Water Development Board (TWDB) stations from the TexMesonet API.

Usage

```
tex_meso_stations(active = NULL, displayed = NULL)
```

Arguments

`active`	Optional logical value used to filter stations by active status. If `NULL` (default), active and inactive stations are returned.
`displayed`	Optional logical value used to filter stations by whether they are displayed by TexMesonet. If `NULL` (default), displayed and hidden stations are returned.

Details

[Experimental]

TexMesonet is managed by the Texas Water Development Board. Its public API provides near-real-time data for TWDB weather and soil observing stations. Review the TexMesonet disclaimer before automated use.

Value

A tibble containing station metadata returned by TexMesonet, including `station_id`, `station_name`, display ID, state, county, latitude, longitude, elevation in feet, active status, and online date.

References

TexMesonet APIs: https://www.texmesonet.org/Apis

TexMesonet disclaimer: https://www.texmesonet.org/About

Examples

```
## Not run:
stations <- tex_meso_stations(active = TRUE)
subset(stations, county == "Blanco")

## End(Not run)
```

`tex_meso_timeseries` *Get recent TexMesonet time-series data*

Description

Retrieves recent time-series observations for a single TWDB TexMesonet station.

Usage

```
tex_meso_timeseries(
  site_id,
  prior_minutes,
  variable = c("all", "temperature", "humidity", "barometric_pressure", "precip",
    "wind_speed")
)
```

Arguments

`site_id`	TexMesonet station ID.
`prior_minutes`	Number of minutes before the current time to retrieve.
`variable`	Data type to retrieve. Use `"all"` for the charting fields endpoint, or one of `"temperature"`, `"humidity"`, `"barometric_pressure"`, `"precip"`, or `"wind_speed"` for the single-variable endpoints.

Details

[Experimental]

TexMesonet's public time-series API retrieves recent observations by station and look-back window. The API may return different fields by station because not all stations measure every parameter.

Value

A tibble of observations with UTC `date_time` values. Single-variable requests return `value` and `date_time` columns and have `"field_name"`, `"station_name"`, `"station_id"`, and `"units"` attributes. Requests with `variable = "all"` return one column per charting field and have `"station_name"` and `"station_id"` attributes.

References

TexMesonet APIs: https://www.texmesonet.org/Apis

Examples

```
## Not run:
tex_meso_timeseries(2, prior_minutes = 60)
tex_meso_timeseries(2, prior_minutes = 60, variable = "temperature")

## End(Not run)
```

---

xCO2_to_pCO2 *Convert CO2 mole fraction to partial pressure*

Description

Converts CO2 mole fraction, xCO2, to the water-vapor-corrected partial pressure, pCO2.

Usage

```
xCO2_to_pCO2(xCO2_ppm, temp_water, atmo_press, press_units, ...)
```

Arguments

xCO2_ppm	Numeric vector. Mole fraction of CO2 in air in parts per million.
temp_water	Numeric vector. Water temperature in degrees Celsius.
atmo_press	Numeric vector. Atmospheric pressure.
press_units	Character string giving the units of atmo_press. See convert_pressure() for accepted pressure units.
...	Additional arguments passed to calc_vapor_press(), such as salinity or method.

Details

The conversion uses:

$$pCO2 = (P_{atm} - P_{H2O}) \cdot xCO2$$

where atmospheric pressure and water vapor pressure are in atm and xCO2 is supplied in parts per million.

Value

Numeric vector of partial pressure of CO2 in microatmospheres.

References

Dickson, A.G., Sabine, C.L., and Christian, J.R. (Eds.) (2007). Guide to best practices for ocean CO2 measurements. PICES Special Publication 3.

Examples

```
xCO2_to_pCO2(
  xCO2_ppm = 420,
  temp_water = 20,
  atmo_press = 101.325,
  press_units = "kPa",
  salinity = 0,
  method = "MIMSY"
)
```

Index