Package 'weathertools'

February 12, 2026

Title Weather and Atmospheric Calculation Utilities

Version 1.0.0

Description Utility functions for computing common weather and atmospheric quantities such as humidity, vapor pressure, and temperature-derived metrics.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Imports Rcpp, data.table, zoo

Suggests crayon

LinkingTo Rcpp, RcppParallel

Contents

`astrocalc` *astrocalc*

Description

astrocalc

Usage

```
astrocalc(
  day,
  month,
  year,
  hour,
  timezone,
  lat,
  lon,
  withinput = FALSE,
  seaorland = "maritime",
  acknowledgment = FALSE
)
```

`avgwdir` *Moving-window average wind direction*

Description

Computes a rolling-mean wind direction using vector (u/v-style) averaging: directions are converted to north/south and east/west components, summed in a rolling window with weights given by wind speed, and converted back to a direction in degrees.

Usage

```
avgwdir(winddirectionsDeg, windspeeds, movingWindow = 10, na.pad = TRUE)
```

Arguments

`winddirectionsDeg`	Numeric vector. Wind direction in degrees (0–360).
`windspeeds`	Numeric vector. Wind speed (same length as `winddirectionsDeg`).
`movingWindow`	Integer. Window size (number of samples) for the rolling sum.
`na.pad`	Logical. If `TRUE`, pads the leading window with `NA` values to preserve input length; if `FALSE`, returns only the fully-defined values.

Details

This function performs a rolling sum on weighted wind components and converts back to a direction via `atan2()`. When `na.pad = TRUE`, the result length equals the input length.

Value

Numeric vector of averaged wind direction degrees in [0, 360).

Examples

```
wd  <- c(350, 10, 15, 20, 25)
wsp <- c(5,   5,  5,  5,  5)
avgwdir(wd, wsp, movingWindow = 3)
```

`calcHI` *Compute Heat Index (fast parallel C++ core)*

Description

Calculates Heat Index (HI) from air temperature and relative humidity. Handles input unit declaration via argument and/or `attr(x, "unit")`. The C++ core computes HI in Fahrenheit; output can be returned in `"degF"` or `"degC"`.

Usage

```
calcHI(
  airTemp,
  relativeHumidity,
  inputunits = NULL,
  outputunits = "degF",
  roundby = 1,
  returnWithUnits = TRUE,
  ignoreattr = FALSE,
  debug = FALSE
)
```

Arguments

`airTemp`	Numeric vector of air temperature. May carry `attr(., "unit")`.
`relativeHumidity`	Numeric vector of RH in percent (0-100).
`inputunits`	Character or `NULL`; one of `"degC"`, `"degF"`, `"K"`. Only needed if `ignoreattr = TRUE` and no attribute is present.
`outputunits`	Character; `"degF"` (default) or `"degC"` for the returned HI.
`roundby`	Integer; decimal places to round the output (default 1).
`returnWithUnits`	Logical; if `TRUE`, sets `attr(result, "unit")` to `outputunits`.
`ignoreattr`	Logical; see Unit handling.
`debug`	Logical; if `TRUE`, emit a brief trace of unit decisions.

Value

Numeric vector of Heat Index in `outputunits`. If `returnWithUnits = TRUE`, the result has `attr(x, "unit") = outputunits`.

Unit handling

- If `ignoreattr = FALSE` (default), the function requires `attr(airTemp, "unit")` and enforces agreement with `inputunits` when provided; otherwise it errors on mismatch/missing attribute.
- If `ignoreattr = TRUE`, the function accepts either the attribute or `inputunits` (but needs at least one).
- Internally, temperature is normalized to `"degF"` before calling the C++ core.

See Also

`unit` for lightweight unit tagging.

Examples

```
# Attribute-driven (degC -> converted internally to degF for core)
ta <- c(30, 35, 40)
attr(ta, "unit") <- "degC"
rh <- c(50, 60, 65)
hiF <- calcHI(ta, rh, outputunits = "degF")
attr(hiF, "unit")  # "degF"

# Explicit input units (no attribute required when ignoreattr = TRUE)
ta2 <- c(86, 95, 104)
hi2 <- calcHI(ta2, rh, inputunits = "degF", outputunits = "degF", ignoreattr = TRUE)
attr(hi2, "unit")  # "degF"

# Ignore attribute and trust supplied units (override a bad attribute)
ta_bad <- ta
attr(ta_bad, "unit") <- "degF"  # wrong on purpose
hi_ok <- calcHI(ta_bad, rh, inputunits = "degC", outputunits = "degF", ignoreattr = TRUE)

# Return plain numeric (no unit attribute)
hi_num <- calcHI(ta, rh, outputunits = "degC", returnWithUnits = FALSE)
attr(hi_num, "unit")  # NULL

# data.table pattern (no {units} required)
if (requireNamespace("data.table", quietly = TRUE)) {
  DT <- data.table::data.table(ta = c(30, 35), rh = c(50, 60))
  attr(DT$ta, "unit") <- "degC"
  DT[, hi := calcHI(ta, rh, outputunits = "degC")]
  attr(DT$hi, "unit")  # "degC"
}
```

calcPres *Sea-level pressure from station pressure*

Description

Converts station pressure (in millibars / hPa) to sea-level pressure using a standard atmosphere approximation and the station elevation.

Usage

```
calcPres(
  pressureMB,
  airTemp,
  elevation,
  inputunits = "degC",
  elevUnits = "m",
  ignoreattr = TRUE,
  quiet = TRUE
)
```

Arguments

pressureMB	Numeric vector. Station pressure in millibars (hPa).
airTemp	Numeric vector. Air temperature at the station. Units set by inputunits.
elevation	Numeric vector or length-1 scalar. Station elevation. Units set by elevUnits.
inputunits	Character scalar. Temperature units: "degC" (default) or "degF".
elevUnits	Character scalar. Elevation units: "m" (default) or "ft".
ignoreattr	Logical.
quiet	Logical.

Details

If elevation is a scalar, it is recycled to match pressureMB.

Value

Numeric vector of sea-level pressure (mb / hPa), same length as pressureMB.

Examples

```
# Example: convert 1000 mb at 20C and 100 m elevation
calcPres(pressureMB = 1000, airTemp = 20, elevation = 100)

# Vectorized
calcPres(pressureMB = c(995, 1002), airTemp = c(15, 18), elevation = 120)
```

`calcRH` *Compute Relative Humidity from Temperature and Dew Point (or VPD)*

Description

Computes RH (%) given air temperature and dew point, or alternatively air temperature and vapor pressure deficit (VPD, kPa). Handles temperature unit declaration via argument and/or `attr(x, "unit")`.

Usage

```
calcRH(
  airTemp,
  dewPoint = NULL,
  inputunits = "degC",
  vpd = NULL,
  ignoreattr = FALSE,
  debug = FALSE
)
```

Arguments

`airTemp`	Numeric vector of air temperature. May carry `attr(., "unit")`.
`dewPoint`	Numeric vector of dew point temperature (optional if `vpd` is supplied). May carry `attr(., "unit")`.
`inputunits`	Character; temperature unit token for `airTemp`/`dewPoint`. One of `"degC"`, `"degF"`, `"K"` (also accepts `"C"`, `"F"`). Defaults to `"degC"`.
`vpd`	Numeric vector of vapor pressure deficit in kPa. If provided, `dewPoint` is ignored.
`ignoreattr`	Logical; if `TRUE`, relax attribute checks and trust `inputunits`.
`debug`	Logical; if `TRUE`, emit a brief trace of unit decisions.

Details

Internally, temperatures are normalized to `"degC"` or `"degF"` depending on `inputunits`, then passed to a C++ routine. When `vpd` is supplied, `dewPoint` is ignored and RH is computed from `airTemp` and `vpd`.

Value

Numeric vector of relative humidity in percent (0-100). No unit attribute is attached.

Unit handling

- Recognized temperature tokens: `"degC"`, `"degF"`, `"K"`.
- `inputunits` applies to `airTemp` (and `dewPoint` if provided).
- If `ignoreattr = FALSE` (default), the function requires unit attributes on provided temperature vectors and enforces agreement with `inputunits` when specified.
- If `inputunits == "K"`, values are normalized to `"degC"` for the core.

See Also

calcTD, unit.

Examples

```
# Using dew point
ta <- c(30, 31); dp <- c(20, 21)
attr(ta, "unit") <- "degC"; attr(dp, "unit") <- "degC"
rh <- calcRH(ta, dp, inputunits = "degC")
rh  # percent

# Using VPD (kPa), ignoring attributes and declaring units
taF <- c(86, 88); vpd <- c(1.5, 2.0)
rh2 <- calcRH(taF, vpd = vpd, inputunits = "degF", ignoreattr = TRUE)

# data.table pattern
if (requireNamespace("data.table", quietly = TRUE)) {
  DT <- data.table::data.table(ta = ta, dp = dp)
  attr(DT$ta, "unit") <- "degC"; attr(DT$dp, "unit") <- "degC"
  DT[, rh := calcRH(ta, dewPoint = dp, inputunits = "degC")]
}
```

---

calcTD *Compute Dew Point Temperature (fast C++ core)*

Description

Calculates dew point from air temperature and relative humidity using a C++ implementation. Inputs can be declared via `inputunits` and/or inferred from `attr(x, "unit")` on `airTemp`. Output is returned in the requested unit and, by default, tagged with a lightweight `"unit"` attribute.

Usage

```
calcTD(
  airTemp,
  relativeHumidity,
  inputunits = "degC",
  outputunits = "degC",
  roundby = 2,
  returnWithUnits = TRUE,
  ignoreattr = FALSE,
  debug = FALSE
)
```

Arguments

`airTemp`	Numeric vector of air temperature. May carry `attr(., "unit")`.
`relativeHumidity`	Numeric vector of RH in percent (0-100).
`inputunits`	Character; temperature units of `airTemp`. One of `"degC"`, `"degF"`, `"K"`. Defaults to `"degC"`.

outputunits	Character; desired dew point units. One of "degC", "degF". Defaults to "degC".
roundby	Integer; number of decimal places to round the result (performed in C++).
returnWithUnits	Logical; if TRUE, sets attr(result, "unit") to outputunits.
ignoreattr	Logical; if TRUE, skip attribute checks and rely on inputunits.
debug	Logical; if TRUE, emit a brief trace of unit decisions.

Value

Numeric vector of dew point in outputunits. If returnWithUnits = TRUE, the result has attr(x, "unit") = outputunits.

Unit handling

- Recognized temperature tokens: "degC", "degF", "K".
- If ignoreattr = FALSE (default), the function reads attr(airTemp, "unit") and enforces agreement with inputunits when provided; otherwise it errors on conflict or missing attribute.
- If ignoreattr = TRUE, the function trusts inputunits (or the attribute if inputunits is missing).
- Internally, the C++ core expects either "degC" or "degF". If inputunits == "K", values are normalized to "degC".

See Also

unit for lightweight unit tagging.

Examples

```
# Simple vector (attribute-driven)
ta <- c(30, 31, 29); attr(ta, "unit") <- "degC"
rh <- c(50, 55, 60)
dpC <- calcTD(ta, rh, outputunits = "degC")
attr(dpC, "unit")  # "degC"

# Override: declare input as Fahrenheit, request Fahrenheit output
taF <- c(86, 88, 84); attr(taF, "unit") <- "degF"
dpF <- calcTD(taF, rh, inputunits = "degF", outputunits = "degF")

# Ignore attribute and trust inputunits
attr(taF, "unit") <- "degC"  # wrong on purpose
dp_ok <- calcTD(taF, rh, inputunits = "degF", outputunits = "degC", ignoreattr = TRUE)

# data.table pattern
if (requireNamespace("data.table", quietly = TRUE)) {
  DT <- data.table::data.table(ta = ta, rh = rh)
  attr(DT$ta, "unit") <- "degC"
  DT[, dp := calcTD(ta, rh, outputunits = "degC")]
  attr(DT$dp, "unit")
}
```

calcWB *Calculate Wet-Bulb Temperature (strict, attribute-aware)*

Description

Computes wet-bulb temperature (WB) from air temperature and relative humidity, using a compiled C++ routine (calcWB_cpp). This version follows your lightweight unit model:

- Inputs are plain numerics with an optional attr(x, "unit").
- By default (ignoreattr = FALSE) a **unit attribute is required** on airTemp (and must match inputunits if you also supply it). Set ignoreattr = TRUE to skip attribute checks and rely solely on inputunits.
- Accepts temperature units "degC", "degF", or "K". When "K" is used, values are converted to "degC" internally for the C++ call.

relativeHumidity may be given as **percent** (0-100) or **fraction** (0-1). Fractions (<= 1.5) are auto-scaled to percent.

Usage

```
calcWB(
  airTemp,
  relativeHumidity,
  inputunits = "degF",
  outputunits = "degF",
  method = NULL,
  ignoreattr = FALSE,
  returnWithUnits = TRUE,
  debug = FALSE
)
```

Arguments

airTemp	Numeric. Air temperature. Provide attr(airTemp, "unit") as "degC", "degF", or "K" unless ignoreattr = TRUE.
relativeHumidity	Numeric. Relative humidity; percent (0-100) or fraction (0-1). Fractions are auto-converted to percent.
inputunits	Character. Temperature unit of airTemp ("degF", "degC", or "K"). Required when ignoreattr = TRUE; if provided together with an attribute, they must agree. Default: "degF".
outputunits	Character. Desired WB output unit ("degF" or "degC"). Default: "degF".
method	Character or NULL. Passed through to calcWB_cpp to select an algorithmic variant (keep NULL for the default).
ignoreattr	Logical. If FALSE (default), require a unit attribute on airTemp and validate against inputunits when provided. If TRUE, skip attribute checks and rely only on inputunits.
returnWithUnits	Logical. If TRUE, tag the numeric result with attr(, "unit") = outputunits. Default: FALSE.
debug	Logical. If TRUE, print a compact summary of normalized inputs.

Value

Numeric vector of wet-bulb temperature in `outputunits`. If `returnWithUnits = TRUE`, the numeric is tagged with `attr(, "unit")`.

Examples

```
## Not run:
# Strict mode (attribute required):
T <- 90; attr(T, "unit") <- "degF"
calcWB(T, 70)                        # returns degF by default
calcWB(T, 70, outputunits = "degC")  # convert to degC

# Kelvin input:
Tk <- 305.15; attr(Tk, "unit") <- "K"
calcWB(Tk, 0.6, outputunits = "degC")                # RH as fraction → auto scaled

# Ignore attributes and use explicit units:
calcWB(32, 40, inputunits = "degC", ignoreattr = TRUE)

## End(Not run)
```

`calcWindchill` *Wind chill (US NWS formula)*

Description

Computes wind chill using the standard U.S. National Weather Service formula.

Usage

```
calcWindchill(airTemp, sfcWind, roundBy = 1)
```

Arguments

`airTemp`	Numeric. Air temperature in degrees Fahrenheit.
`sfcWind`	Numeric. Wind speed in mph.
`roundBy`	Integer. Number of decimal places to round to (default 1).

Value

Numeric vector of wind chill values.

Examples

```
calcWindchill(airTemp = 30, sfcWind = 10)
calcWindchill(airTemp = c(30, 25), sfcWind = c(10, 15), roundBy = 0)
```

convert_units *Convert numeric vectors between a fixed set of common meteorological units*

Description

`convert_units()` performs fast, vectorized unit conversion for a **fixed** set of unit pairs used in this project. It is intentionally lightweight and does **not** depend on the **units** package or any unit classes.

Units are normalized via the internal helper `.norm_u()` (e.g., `"C"`, `"degC"`, `"celsius"` → `"degC"`). If `from` or `to` are `NULL`, unknown, or identical after normalization, the input is returned unchanged.

Usage

```
convert_units(x, from, to)
```

Arguments

`x`	Numeric vector. Values to convert. Coercion is not performed; callers should ensure `x` is numeric.
`from`	Character scalar. Source unit token (e.g. `"K"`, `"degC"`, `"mph"`, `"Pa"`). Normalized internally by `.norm_u()`.
`to`	Character scalar. Target unit token (e.g. `"degC"`, `"m/s"`, `"hPa"`). Normalized internally by `.norm_u()`.

Details

Supported conversions include (not exhaustive):

- Temperature: `degC` to/from `degF`; `degC` to/from `K`; `degF` to/from `K`
- Pressure: `Pa` to/from `hPa`
- Wind speed: `mph` to/from `m/s`; `kt` to/from `m/s`
- Distance: `feet` to/from `m`; `mile` to/from `km`
- Water-equivalent flux/accumulation: `kg/m^2/s` to/from `mm/h`; `kg/m^2` to/from `mm`; plus `mm` to/from `in` and `kg/m^2` to/from `in` (using 1 mm water = 1 kg/m^2)
- Identity passes for common tokens (e.g. `"%"` and `"W/m^2"`)

If a requested conversion is not implemented, the function errors with a message like `"No converter for degC -> degF"`.

Value

A numeric vector of the same length as `x`, converted to `to`. If no conversion is performed, returns `x` unchanged.

Examples

```
# Temperature
convert_units(c(0, 20, 30), "degC", "degF")
convert_units(c(32, 68), "degF", "degC")
convert_units(c(300, 310), "K", "degC")

# Wind speed
convert_units(c(10, 25), "mph", "m/s")
convert_units(c(5, 10), "m/s", "mph")
convert_units(c(10, 20), "kt", "m/s")

# Pressure
convert_units(c(101325, 100800), "Pa", "hPa")

# Precip/water equivalents
convert_units(c(1.5, 3), "mm", "in")
convert_units(c(2e-4, 5e-4), "kg/m^2/s", "mm/h")
```

---

ctof *Convert Celsius to Fahrenheit (lightweight unit attributes)*

Description

Converts temperatures expressed in Celsius to Fahrenheit using lightweight unit handling via `attr(x, "unit")`. This function does not use the **units** package.

Usage

```
ctof(c, ignoreattr = FALSE, quiet = TRUE, return_original_as_attr = TRUE, ...)
```

Arguments

`c`	Numeric vector of temperatures in Celsius. May carry `attr(., "unit")`.
`ignoreattr`	Logical; if `FALSE`, require and validate `attr(c, "unit")` is Celsius.
`quiet`	Logical; if `FALSE`, emit a short message about the conversion.
`return_original_as_attr`	Logical; if `TRUE`, attach `attr(out, "original")` containing the original numeric input (assumed/validated Celsius).
`...`	Reserved for backward compatibility (ignored).

Value

Numeric vector of temperatures in Fahrenheit. The result carries `attr(out, "unit") = "degF"`. If `return_original_as_attr = TRUE`, `attr(out, "original")` holds the original numeric Celsius values.

Unit handling

- If `ignoreattr = FALSE` (default), `c` must have `attr(c, "unit")` that resolves to `"degC"`; otherwise an error is thrown.
- If `ignoreattr = TRUE`, the attribute is ignored and the input is assumed to be Celsius.
- Output is a numeric vector in Fahrenheit with `attr(out, "unit") = "degF"`.

Examples

```
x <- c(0, 25, 30)
attr(x, "unit") <- "degC"
y <- ctof(x)
y
attr(y, "unit")  # "degF"
attr(y, "original")

# Ignore missing attribute (assume Celsius)
ctof(c(0, 25), ignoreattr = TRUE)
```

`degtoradian` *Degrees to radians*

Description

Degrees to radians

Usage

```
degtoradian(deg)
```

Arguments

`deg`	Numeric vector of angles in degrees.

Value

Numeric vector of angles in radians.

`ftoc` *Convert Fahrenheit to Celsius (lightweight unit attributes)*

Description

Converts temperatures expressed in Fahrenheit to Celsius using lightweight unit handling via `attr(x, "unit")`. This function does not use the **units** package.

Usage

```
ftoc(f, ignoreattr = FALSE, quiet = TRUE, return_original_as_attr = TRUE, ...)
```

Arguments

`f`	Numeric vector of temperatures in Fahrenheit. May carry `attr(., "unit")`.
`ignoreattr`	Logical; if `FALSE`, require and validate `attr(f, "unit")` is Fahrenheit.
`quiet`	Logical; if `FALSE`, emit a short message about the conversion.
`return_original_as_attr`	Logical; if `TRUE`, attach `attr(out, "original")` containing the original numeric input (assumed/validated Fahrenheit).
`...`	Reserved for backward compatibility (ignored).

Value

Numeric vector of temperatures in Celsius. The result carries `attr(out, "unit") = "degC"`. If `return_original_as_attr = TRUE`, `attr(out, "original")` holds the original numeric Fahrenheit values.

Unit handling

- If `ignoreattr = FALSE` (default), `f` must have `attr(f, "unit")` that resolves to `"degF"`; otherwise an error is thrown.
- If `ignoreattr = TRUE`, the attribute is ignored and the input is assumed to be Fahrenheit.
- Output is a numeric vector in Celsius with `attr(out, "unit") = "degC"`.

Examples

```
x <- c(32, 68, 77)
attr(x, "unit") <- "degF"
y <- ftoc(x)
y
attr(y, "unit")  # "degC"

# Ignore missing attribute (assume degF)
ftoc(c(32, 212), ignoreattr = TRUE)
```

---

intomb *Inches of mercury to millibars*

Description

Inches of mercury to millibars

Usage

```
intomb(presin)
```

Arguments

`presin`	Numeric vector of pressure in inches of mercury (inHg).

Value

Numeric vector of pressure in millibars (hPa).

mphtoms *Convert miles per hour to meters per second (lightweight unit attributes)*

Description

Converts wind speed from "mph" to "m/s" using lightweight unit handling via attr(x, "unit"). This function does not use the **units** package.

Usage

```
mphtoms(
  thedata,
  ignoreattr = FALSE,
  quiet = TRUE,
  return_original_as_attr = TRUE,
  ...
)
```

Arguments

thedata	Numeric vector of wind speed in miles per hour. May carry attr(., "unit").
ignoreattr	Logical; if FALSE, require and validate attr(thedata, "unit") is "mph".
quiet	Logical; if FALSE, emit a short message about the conversion.
return_original_as_attr	Logical; if TRUE, attach attr(out, "original") containing the original numeric input (assumed/validated "mph").
...	Reserved for backward compatibility (ignored).

Value

Numeric vector of wind speed in meters per second. The result carries attr(out, "unit") = "m/s". If return_original_as_attr = TRUE, attr(out, "original") holds the original numeric miles-per-hour values.

Unit handling

- If ignoreattr = FALSE (default), thedata must have attr(thedata, "unit") that resolves to "mph"; otherwise an error is thrown.
- If ignoreattr = TRUE, the attribute is ignored and the input is assumed to be "mph".
- Output is a numeric vector in "m/s" with attr(out, "unit") = "m/s".

Examples

```
x <- c(0, 10, 25)
attr(x, "unit") <- "mph"
y <- mphtoms(x)
y
attr(y, "unit")  # "m/s"
attr(y, "original")
```

```
# Ignore missing attribute (assume mph)
mphtoms(c(5, 15), ignoreattr = TRUE)
```

`mstomh` *Convert meters per second to miles per hour (lightweight unit attributes)*

Description

Converts wind speed from `"m/s"` to `"mph"` using lightweight unit handling via `attr(x, "unit")`. This function does not use the **units** package.

Usage

```
mstomh(
  windspeed,
  ignoreattr = FALSE,
  quiet = TRUE,
  return_original_as_attr = TRUE,
  ...
)
```

Arguments

`windspeed`	Numeric vector of wind speed in meters per second. May carry `attr(., "unit")`.
`ignoreattr`	Logical; if `FALSE`, require and validate `attr(windspeed, "unit")` is `"m/s"`.
`quiet`	Logical; if `FALSE`, emit a short message about the conversion.
`return_original_as_attr`	Logical; if `TRUE`, attach `attr(out, "original")` containing the original numeric input (assumed/validated `"m/s"`).
`...`	Reserved for backward compatibility (ignored).

Value

Numeric vector of wind speed in miles per hour. The result carries `attr(out, "unit") = "mph"`. If `return_original_as_attr = TRUE`, `attr(out, "original")` holds the original numeric meters-per-second values.

Unit handling

- If `ignoreattr = FALSE` (default), `windspeed` must have `attr(windspeed, "unit")` that resolves to `"m/s"`; otherwise an error is thrown.
- If `ignoreattr = TRUE`, the attribute is ignored and the input is assumed to be `"m/s"`.
- Output is a numeric vector in `"mph"` with `attr(out, "unit") = "mph"`.

Examples

```
x <- c(0, 5, 10)
attr(x, "unit") <- "m/s"
y <- mstomph(x)
y
attr(y, "unit")  # "mph"

# Ignore missing attribute (assume m/s)
mstomph(c(5, 7), ignoreattr = TRUE)
```

`mstomph` *Convert meters per second to miles per hour (lightweight unit attributes)*

Description

Converts wind speed from `"m/s"` to `"mph"` using lightweight unit handling via `attr(x, "unit")`. This function does not use the **units** package.

Usage

```
mstomph(
  thedata,
  ignoreattr = FALSE,
  quiet = TRUE,
  return_original_as_attr = TRUE,
  ...
)
```

Arguments

`thedata`	Numeric vector of wind speed in meters per second. May carry `attr(., "unit")`.
`ignoreattr`	Logical; if `FALSE`, require and validate `attr(thedata, "unit")` is `"m/s"`.
`quiet`	Logical; if `FALSE`, emit a short message about the conversion.
`return_original_as_attr`	Logical; if `TRUE`, attach `attr(out, "original")` containing the original numeric input (assumed/validated `"m/s"`).
`...`	Reserved for backward compatibility (ignored).

Value

Numeric vector of wind speed in miles per hour. The result carries `attr(out, "unit") = "mph"`. If `return_original_as_attr = TRUE`, `attr(out, "original")` holds the original numeric meters-per-second values.

Unit handling

- If `ignoreattr = FALSE` (default), `thedata` must have `attr(thedata, "unit")` that resolves to `"m/s"`; otherwise an error is thrown.
- If `ignoreattr = TRUE`, the attribute is ignored and the input is assumed to be `"m/s"`.
- Output is a numeric vector in `"mph"` with `attr(out, "unit") = "mph"`.

Examples

```
x <- c(0, 5, 10)
attr(x, "unit") <- "m/s"
y <- mstomph(x)
y
attr(y, "unit")  # "mph"
attr(y, "original")

# Ignore missing attribute (assume m/s)
mstomph(c(5, 7), ignoreattr = TRUE)
```

---

stationpressure *Estimate station pressure from sea-level pressure and elevation*

Description

Estimate station pressure from sea-level pressure and elevation

Usage

```
stationpressure(
  pressureMB,
  airTemp,
  elevation,
  airTempUnits = "C",
  elevUnits = "m"
)
```

Arguments

pressureMB	Numeric vector of pressure (typically sea-level) in millibars/hPa.
airTemp	Numeric vector of air temperature.
elevation	Numeric vector of station elevation.
airTempUnits	Character temperature units for airTemp (e.g. "degC" or "degF").
elevUnits	Character elevation units for elevation (e.g. "m" or "feet").

Value

Numeric vector of estimated station pressure (typically in millibars/hPa).

tzone *Lightweight time zone getter/setter for POSIXt*

Description

Convenience helpers to read or set the "tzone" attribute on POSIX date-times. Use "Area/Location" to relabel (same instant), or append "|force" to re-interpret clock times as local in the new zone (epoch seconds change).

Usage

```
tzone(x)

tzone(x) <- value
```

Arguments

x	A POSIXct or POSIXlt vector.
value	Character scalar time zone in IANA/Olson form (see OlsonNames()), optionally annotated with "|force" or "; mode=force".

Details

- tzone(x) returns the current time zone tag (character) or "".
- tzone(x) <- value sets the time zone. By default it relabels only. If value indicates "force", the same printed wall time is re-parsed in the new zone (epoch seconds change).

Accepted value forms:

- "America/New_York" — relabel only
- "America/New_York|force" — force reinterpretation
- "America/New_York; mode=force" — same as above

Value

tzone(x) returns the current tag (character) or "". The replacement form returns the modified vector.

Examples

```
x <- as.POSIXct("2024-07-01 12:00:00", tz = "UTC")
tzone(x)

# Relabel only (same instant)
tzone(x) <- "America/New_York"
format(x, tz = tzone(x))

# Force reinterpretation (epoch seconds change)
y <- as.POSIXct("2024-07-01 12:00:00", tz = "UTC")
tzone(y) <- "America/New_York|force"
y
```

`unit` *Lightweight unit getter/setter (conversion + optional rounding)*

Description

Provides a simple way to read and set a `"unit"` attribute on numeric vectors. The replacement form (`unit(x) <- value`) can also convert values between supported units (handled by the package-internal `weathertools::convert_units()`) and optionally round the result.

Usage

```
unit(x)

unit(x) <- value
```

Arguments

`x`	A numeric vector (or column) to query or tag with a `"unit"` attribute.
`value`	Character scalar describing the assignment. See the accepted forms above. Examples: `"degF"`, `"degC -> degF"`, `"degC|degF|1"`, `"degF (2)"`, `"degF; round=1"`.

Details

Accepted assignment forms for `value`:

- `"degF"` — set/convert to `degF` using the current attribute as the source (if present).
- `"degC -> degF"` — explicitly convert from `degC` to `degF`.
- `"degC|degF"` — shorthand for the arrow form above.
- Optional rounding may be requested (highest precedence first): `"...; round=1"`, `"...; digits=1"`, trailing `"|1"`, or trailing `"(1)"`; e.g., `"degC|degF|1"` or `"degF (2)"`.

If an explicit source unit is provided on the left side (e.g., `"degC -> degF"`), it is treated as authoritative. To prevent accidental re-interpretation of already-tagged data, a source-mismatch policy is applied:

- `options(jj.unit.on_src_mismatch = "warn_noop")` (default) — warn and do nothing when *declared source* differs from `attr(x,"unit")`.
- `"error"` — stop with an error.
- `"convert"` — trust the declared source and convert anyway (original permissive behavior).

Rounding is applied *after* conversion. Global default rounding can be set via `options(jj.unit.round_digits = K)`. This default is only applied when a conversion actually occurred; inline digits (`round= / digits= / |K / (K)`) always apply. When rounding occurs, the setter also records `attr(x, "unit_digits") = K`.

To avoid silently mislabeling temperatures, you may forbid "tag-only" temperature assignments (no known source) by enabling: `options(jj.unit.disallow_temp_tag_only = TRUE)`. With this option set, attempting `unit(x) <- "degF"` on an untagged vector will error; use `"src|dst"` instead.

Value

- `unit(x)` returns the current unit attribute (character scalar) or `NULL`.
- `unit(x) <- value` returns the modified vector `x`, with values converted if needed, optionally rounded, and `attr(x, "unit")` set to the target unit. When rounding is applied, `attr(x, "unit_digits")` is also set.

Conversions

The actual numeric conversions are performed by the internal `weathertools::convert_units(x, from, to)`. Typical pairs supported in this package include temperature (`degC`, `degF`, `K`/`degK`), pressure (`hPa`, `Pa`), wind speed (`m/s`, `mph`, `kt`), precipitation depth (`kg/m^2`, `mm`, `in`), radiation (`W/m^2`), and relative humidity forms (where supported by your map).

Examples

```
x <- c(25, 26, 27)
unit(x) <- "degC"        # tag as degC
unit(x)                  # "degC"

# Convert using current attribute as the source
unit(x) <- "degF"        # C -> F (if attr is "degC")

# Explicit source -> target
unit(x) <- "degF -> degC"

# Shorthand with rounding
unit(x) <- "degC|degF|1"   # C -> F, then round(., 1)
unit(x) <- "degF (2)"      # tag/convert to F, then round(., 2)

# Safer everyday pattern (idempotent):
y <- c(0, 5, 10); unit(y) <- "degC"
unit(y) <- "degF"          # converts once; calling again is a no-op

# data.table in-place usage
if (requireNamespace("data.table", quietly = TRUE)) {
  library(data.table)
  DT <- data.table(ta = c(25, 26, 27))
  unit(DT$ta) <- "degC"
  DT[, ta := { unit(ta) <- "degC -> degF; round=1"; ta }]
  unit(DT$ta)                    # "degF"
  attr(DT$ta, "unit_digits")     # 1
}

# Policies (optional):
# options(jj.unit.on_src_mismatch = "warn_noop")  # default
# options(jj.unit.on_src_mismatch = "error")
# options(jj.unit.on_src_mismatch = "convert")
#
# options(jj.unit.round_digits = 1L)              # global rounding (after conversions)
# options(jj.unit.disallow_temp_tag_only = TRUE)  # forbid tag-only for temperatures
```

unitConvertRound *Convert and round a vector between units (lightweight conversions)*

Description

Converts numeric vectors from one unit to another using weathertools' internal conversion table (see `.convert_units()`) and rounds the result to a specified number of decimal places. This function does not use the **units** package and only supports unit pairs present in the internal conversion table.

Usage

```
unitConvertRound(x, from, to, digits = 2, strip = FALSE)
```

Arguments

`x`	Numeric vector (optionally carrying `attr(x, "unit")`).
`from`	Character. Original unit of `x` (e.g., `"degC"`, `"m/s"`).
`to`	Character. Desired target unit (e.g., `"degF"`, `"mph"`).
`digits`	Integer. Number of decimal places to round the converted values (default `2`).
`strip`	Logical. If `TRUE`, remove the unit attribute from the result (default `FALSE`).

Details

This function calls `.convert_units(x, from, to)` and then rounds the result. If a requested conversion is not present in the internal table, an error is thrown.

If `x` already has `attr(x, "unit")` and `from` is `NULL`, the attribute is used as the source unit. If both are provided and disagree, an error is thrown unless `strip = TRUE` is used to explicitly drop unit tagging on output (conversion still requires a valid `from`).

Value

A numeric vector of converted values, rounded to `digits`. By default, `attr(out, "unit")` is set to the canonical target unit `to`.

Examples

```
# Convert temperature from degC to degF
wbgt_c <- c(20, 25, 30)
unitConvertRound(wbgt_c, from = "degC", to = "degF", digits = 1)

# Convert wind speed from m/s to mph
speed_ms <- c(5, 10, 15)
unitConvertRound(speed_ms, from = "m/s", to = "mph", digits = 2)

# Attribute-driven source unit (from inferred)
x <- c(0, 10)
attr(x, "unit") <- "degC"
unitConvertRound(x, from = NULL, to = "degF", digits = 1)
```

uv2wdws *Convert u/v wind components to direction and speed*

Description

Converts u (zonal, positive eastward) and v (meridional, positive northward) wind components into meteorological wind direction (degrees from which the wind is blowing) and wind speed.

Usage

```
uv2wdws(u, v)
```

Arguments

u	Numeric vector. Zonal wind component.
v	Numeric vector. Meridional wind component.

Value

A numeric matrix with two columns: `wd` (wind direction degrees, [0, 360)) and `ws` (wind speed, same units as u/v).

Examples

```
uv2wdws(u = c(1, 0, -1), v = c(0, 1, 0))
```

winddeg *Wind direction degrees to 16-point compass labels*

Description

Converts wind direction degrees into standard 16-point compass labels: N, NNE, NE, ENE, E, ESE, SE, SSE, S, SSW, SW, WSW, W, WNW, NW, NNW.

Usage

```
winddeg(windDeg)
```

Arguments

windDeg	Numeric vector of wind directions in degrees.

Value

Character vector of compass labels (same length as `windDeg`).

Examples

```
winddeg(c(0, 20, 45, 90, 200, 359.9))
```

windRun *Wind run*

Description

Wind run

Usage

```
windRun(windSpd, archiveInterval = 1, roundBy = 5)
```

Arguments

windSpd	Numeric vector of wind speed values.
archiveInterval	Numeric or integer. Time interval represented by each wind speed value (document units: seconds/minutes/hours, whichever you use).
roundBy	Integer; decimal places to round output.

Value

Numeric vector of wind run (document units).

wx.units *Harmonize/standardize weather fields across feeds (rename + units, in-place)*

Description

`wx.units()` makes a heterogeneous weather feed look consistent: it (optionally) **renames** provider-specific columns to your canonical names and **unit-converts** them to canonical target units — *without* relying on heavy unit classes. It modifies the input `data.table` **by reference** and also writes a human-friendly attribute `attr(x[[col]], "unit")` on each harmonized column.

The function is strict by default:

- It will only infer *source* units from the **original provider column name** (e.g. `TMP_2m_K` → `"K"`).
- It never "guesses". If a column's source unit cannot be determined from the original name or an explicit override (or an existing `"unit"` attribute), it errors unless `stop_if_unknown = FALSE`.

Extras:

- If `wind10m` is missing but `ugrd10m` and `vgrd10m` exist, the function derives wind speed/direction (`wind10m`, `WDIR`).
- If `solar` is absent but `dswrf` exists, it creates `solar := dswrf`.
- Optional runtime verification catches common scale mistakes (e.g. K→°C didn't actually drop by ~273).

Usage

```
wx.units(
  dt,
  rename_map = NULL,
  src_override = NULL,
  target_override = NULL,
  stop_if_unknown = FALSE,
  debug = FALSE
)
```

Arguments

dt	A data.table. Modified in place.
rename_map	Optional named character vector mapping **provider names** to **canonical names** (e.g. c("TMP_2m_K" = "ta", "DSWRF_surface_Wm^2" = "dswrf")). If NULL or length-0, renaming is skipped; unit logic still applies via src_override / existing attr(, "unit").
src_override	Optional named list mapping **canonical names** to *source* units, e.g. list(ta = "K", td = "K", wind10m = "m/s"). Precedence: src_override > detection from original provider names (via rename_map) > existing attr(dt[[col]], "unit"). Nothing is guessed.
target_override	Optional named list mapping **canonical names** to *target* units (overrides the built-in canonical table). Example: list(wind10m = "mph", pres = "Pa").
stop_if_unknown	Logical (default TRUE). If TRUE, error when any column's source unit is unknown. If FALSE, those columns are left untouched (but will not be re-tagged).
debug	Logical. If TRUE, prints a compact conversion report (variable, from, to, sample before/after).

Details

Canonical schema & units:

The default canonical names and target units used by this function are:

- Temperatures: ta, ta_sfc, td → **degC**
- Wind: wind10m, ugrd10m, vgrd10m, GUST → **m/s**; WDIR → **deg**
- Pressure: pres, pres_cloudtop, pres_0Cisotherm, pres_tropopause_freezing → **hPa**
- Radiation: dswrf, solar → **W/m^2**
- Precipitation: prate → **mm/h**, apcp_sfc → **mm**
- Clouds: tcdc, tcdc_bl, lcdc, mcdc, hcdc → **%**
- Visibility & ceiling: VIS → **km**, ceiling → **m**
- Reflectivity/lightning: REFC → **dB**, LTNG, LTNG2 → **1/m^2/s**
- Satellite brightness temps: SBT123, SBT124, SBT113, SBT114 → **K**

You can override any of these targets via target_override.

Where source units come from:

1. **From** rename_map **original names** (preferred when renaming):
   - Suffix patterns like "_K", "_ms", "_Pa", "_knots", "_feet", "_statutemile", "percent", "_degtrue", "_Wm^2", "_kgm^2s", "_kgm^2" are mapped deterministically to units.

2. **From** `src_override` if you know the input unit and either:
   - You aren't renaming, *or*
   - The provider naming is inconsistent.
3. **From existing attribute** `attr(dt[[col]], "unit")` if present.

If none of the above provide a unit for a target variable, the column is considered unknown; see `stop_if_unknown`.

Conversions supported (numeric only):

The function implements a fixed set of conversions used in this project:

- `K <-> degC`, `degC <-> degF`
- `Pa <-> hPa`
- `mph <-> m/s`, `kt → m/s`
- `feet → m`, `mile → km`
- `kg/m^2/s → mm/h`, `kg/m^2 → mm` (water equiv.)
- Identity passes (e.g., `%`, `deg`, `W/m^2`, `dB`, `1/m^2/s`, `m/s`)

Add more as needed inside the function's internal converter.

Side effects & performance:

- **By reference**: `dt` is modified in place; the function returns `dt` invisibly for pipe friendliness.
- **Attributes**: after harmonization, each target column gets `attr(, "unit")` set to the target unit for easy introspection/logging.
- **Vectorized**: all conversions are per-column numeric operations — no row loops and no `units` S3 overhead.

Derivations & aliases:

- If `wind10m` is missing but both `ugrd10m` and `vgrd10m` exist, the function creates `wind10m` (m/s) and `WDIR` (deg).
- If `solar` is absent and `dswrf` exists, `solar := dswrf` (W/m^2) is added.

Error handling:

- Unknown source units → error when `stop_if_unknown = TRUE`.
- Runtime verification checks (e.g., "still looks like Kelvin after K→°C") throw informative errors to catch scale mistakes early.

Value

Invisibly returns `dt` (same reference), with:

- Optional **renaming** applied,
- **Numeric** values converted to canonical target units,
- `attr(, "unit")` set per harmonized column,
- Potentially **derived** `wind10m`, `WDIR`, and **aliased** `solar`.

Canonical targets (quick reference)

See **Details**. Override with `target_override`.

Typical usage patterns

1) Provider → Canonical (rename + units):

```
library(data.table)
dt <- data.table(
  TMP_2m_K               = c(298.15, 300.15),
  DPT_2m_K               = c(293.15, 295.15),
  UGRD_10m_ms            = c( 2.0,  3.5),
  VGRD_10m_ms            = c(-1.0, -2.0),
  DSWRF_surface_Wm^2     = c(500, 750),
  PRES_surface_Pa        = c(101325, 100800),
  TCDC_percent           = c(40, 75)
)

rename_map <- c(
  "TMP_2m_K"           = "ta",
  "DPT_2m_K"           = "td",
  "UGRD_10m_ms"        = "ugrd10m",
  "VGRD_10m_ms"        = "vgrd10m",
  "DSWRF_surface_Wm^2" = "dswrf",
  "PRES_surface_Pa"    = "pres",
  "TCDC_percent"       = "tcdc"
)

wx.units(dt, rename_map, debug = TRUE)
# dt now has ta/td in degC, ugrd/vgrd in m/s, wind10m+WDIR derived, pres in hPa, dswrf in W/m^2,
# and attr(, "unit") set on each target column.
```

2) Unit harmonization only (no renaming):

```
# Suppose your feed already uses canonical names but mixed units:
dt <- data.table(ta = c(297, 300), td = c(290, 293), wind10m = c(12, 8))
attr(dt$ta, "unit") <- "K"     # mark current units
attr(dt$td, "unit") <- "K"
attr(dt$wind10m, "unit") <- "mph"

spec_in  <- list(ta = "K", td = "K", wind10m = "mph")
spec_out <- list(ta = "degC", td = "degC", wind10m = "m/s")

wx.units(
  dt,
  rename_map      = character(),   # skip renaming
  src_override    = spec_in,
  target_override = spec_out,
  debug           = TRUE
)
```

3) Overriding targets (e.g., store wind as mph):

```
dt <- data.table(wind10m = c(5, 8))  # m/s by provider contract
attr(dt$wind10m, "unit") <- "m/s"
wx.units(dt,
  rename_map      = character(),
  target_override = list(wind10m = "mph"),
```

```
  debug            = TRUE
)
```

4) Tolerant mode for partially specified inputs:

```
dt <- data.table(ta = c(295, 300), foo = 1:2) # ta has no unit info
# This will error by default; set stop_if_unknown = FALSE to skip unknowns.
wx.units(
  dt,
  rename_map       = character(),
  src_override     = list(ta = "K"),
  stop_if_unknown  = FALSE
)
```

See Also

- Your package's bias pipeline and ingestion helpers that call this function.
- `data.table::setattr()` for inspecting the `"unit"` tag.

Examples

```
## Not run:
# See the sections above for runnable examples.

## End(Not run)
```

Index