Package ‘myClim’

October 21, 2024

Type Package

Title Microclimatic Data Processing

Version 1.3.0

URL http://labgis.ibot.cas.cz/myclim/index.html,
https://github.com/ibot-geoecology/myClim

Description Handling the microclimatic data in R. The 'myClim' workflow begins
at the reading data primary from microclimatic dataloggers,
but can be also reading of meteorological station data from files.
Cleaning time step, time zone settings and metadata collecting is the next step of the work flow.
With 'myClim' tools one can crop, join, downscale, and convert microclimatic data formats, sort them into localities,
request descriptive characteristics and compute microclimatic variables.
Handy plotting functions are provided with smart defaults.

License GPL (>= 2)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 3.5)

Imports stringr, lubridate, tibble, dplyr, purrr, tidyr, ggplot2,
ggforce, viridis, data.table, plotly, zoo, methods, vroom,
progress

Additional_repositories https://ibot-geoecology.github.io/drat

Suggests rmarkdown, knitr, kableExtra, rTubeDB, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Matěj Man [aut],
Vojtěch Kalčík [aut, cre],
Martin Macek [aut],
Josef Brůna [aut],

Lucia Hederová [aut],
Jan Wild [aut],
Martin Kopecký [aut],
Institute of Botany of the Czech Academy of Sciences [cph]

Maintainer Vojtěch Kalčík <Vojtech.Kalcik@ibot.cas.cz>

Repository CRAN

Date/Publication 2024-10-21 12:21:08 UTC

Contents

`length.myClimList` *Length function for myClim object*

Description

Function return number of localities.

Usage

```
## S3 method for class 'myClimList'
length(x, ...)
```

Arguments

`x`	myClim object see myClim-package
`...`	other parameters from function length

Examples

```
length(mc_data_example_agg)
```

mc_agg *Aggregate data by function*

Description

mc_agg has two basic uses:

- aggregate (upscale) time step of microclimatic records with specified function (e. g. 15 min records to daily mean);
- convert myClim object from Raw-format to Agg-format see myClim-package without time-series modification, this behavior appears when `fun=NULL`, `period=NULL`.

Usage

```
mc_agg(
  data,
  fun = NULL,
  period = NULL,
  use_utc = TRUE,
  percentiles = NULL,
  min_coverage = 1,
  custom_start = NULL,
  custom_end = NULL,
  custom_functions = NULL
)
```

Arguments

`data` cleaned myClim object in Raw-format: output of `mc_prep_clean()` or Agg-format as it is allowed to aggregate data multiple times.

`fun` aggregation function; one of (`"min"`, `"max"`, `"mean"`, `"percentile"`, `"sum"`, `"range"`, `"count"`, `"coverage"`) and functions defined in `custom_functions`. See details of custom_functions argument. Can be single function name, character vector of function names or named list of vector function names. Named list of functions allows apply different function(s) to different sensors e.g. `list(TMS_T1=c("max", "min"), TMS_T2="mean", TMS_T3_GDD="sum")` if NULL records are not aggregated, but myClim object is only converted to Agg-format without modifing time-series. See details.

`period` Time period for aggregation - same as breaks in cut.POSIXt, e.g. (`"hour"`, `"day"`, `"month"`); if NULL then no aggregation

There are special periods `"all"` and `"custom"`. Period `"all"` returning single value for each sensor based on function applied across all records within the sensor. Period `"custom"` aggregates data in yearly cycle. You can aggregate e.g. water year, vegetation season etc. by providing start, end datetime. See `custom_start` and `custom_end` parameters. The output of special periods "all" and "custom"are not allowed to be aggregated again in mc_agg() function, regardless multiple aggregations are allowed in general.

Start day of week is Monday.

use_utc
: default TRUE using UTC time, if set FALSE, the time is shifted by offset if available in locality metadata. Shift can be e.g. to solar time mc_prep_solar_tz() or political time with custom offset mc_prep_meta_locality()). Non-UTC time can by used only for aggregation of the data with period shorter than day (seconds, minutes, hours) into period day and longer.

percentiles
: vector of percentile numbers; numbers are from range 0-100; each specified percentile number generate new virtual sensor, see details

min_coverage
: value from range 0-1 (default 1); the threshold specifying how many missing values can you accept within aggregation period. e.g. when aggregating from 15 min to monthly mean and set `min_coverage=1` then a single NA value within the specific month cause monthly mean = NA. When `min_coverage=0.9` then you will get your monthly mean in case there are no more than 10 % missing values, if there were more than 10% you will get NA. Ignored for functions `count` and `coverage`

custom_start
: date of start, only use for `custom` period (default NULL); Character in format "mm-dd" or "mm-dd H:MM" recycled in yearly cycle for time-series longer than 1 year.

custom_end
: date of end only use for `custom` period (default NULL); If NULL then calculates in year cycle ending on `custom_start` next year. (useful e.g. for hydrological year) When custom_end is provided, then data out of range `custom_start-custom_end` are ignored. Character in format "mm-dd" or "mm-dd H:MM". `custom_end` row (the last record) is not included. I.e.complete daily data from year 2020 ends in 2021-01-01 `custom_end="01-01"`.

custom_functions
: user define one or more functions in format `list(function_name=function(values){...})`; then you will feed function_name(s) you defined to the `fun` parameter. e.g. `custom_functions = list(positive_count=function(x){length(x[x>0])})`, `fun="positive_count"`,

Details

Any output of mc_agg is in Agg-format. That means the hierarchical level of logger is removed (Locality<-Logger<-Sensor<-Record), and all microclimatic records within the sensors are on the level of locality (Locality<-Sensor<-Record). See myClim-package.

In case `mc_agg()` is used only for conversion from Raw-format to Agg-format (`fun=NULL, period=NULL`) then microclimatic records are not modified. Equal step in all sensors is required for conversion from Raw-format to Agg-format, otherwise period must be specified.

When fun and period are specified, microclimatic records are aggregated based on a selected function into a specified period. The name of the aggregated variable will contain also the name of the function used for the aggregation (e.g. TMS_T1_mean). Aggregated time step is named after the first time step of selected period i.e. day = c(2022-12-29 00:00, 2022-12-30 00:00...); week = c(2022-12-19 00:00, 2022-12-28 00:00...); month = c(2022-11-01 00:00, 2022-12-01 00:00...); year = c(2021-01-01 00:00, 2022-01-01 00:00...). When first or last period is incomplete in original data, the incomplete part is extended with NA values to match specified period. For example, when you want to aggregate time-series to monthly mean, but your time-series starts on January 15 ending December 20, myClim will extend the time-series to start on January 1 and end on December 31. If you want to still use the data from the aggregation periods with not complete data coverage, you can adjust the parameter `min_coverage`.

Empty sensors with no records are excluded. `mc_agg()` return NA for empty vector except from `fun=count` which returns 0. When aggregation functions are provided as vector or list e.g. c(mean, min, max), than they are all applied to all the sensors and multiple results are returned from each sensors. When named list (names are the sensor ids) of functions is provided then `mc_agg()` apply specific functions to the specific sensors based on the named list `list(TMS_T1=c("max", "min"), TMS_T2="mean")`. mc_agg returns new sensors on the localities putting aggregation function in its name (TMS_T1 -> TMS_T1_max), despite sensor names contains aggregation function, sensor_id stays the same as before aggregation in sensor metadata (e.g. TMS_T1 -> TMS_T1). Sensors created with functions `min, max, mean, percentile, sum, range` keeps identical sensor_id and value_type as original input sensors. When function `sum` is applied on `logical` sensor (e.g. snow as TRUE, FALSE) the output is `integer` i.e. number of `TRUE` values.

Sensors created with functions `count` has sensor_id `count` and value_type `integer`, function `coverage` has sensor_id `coverage` and value_type `real`

If the myClim object contains any states (tags) table, such as error tags or quality tags, the datetime defining the start and end of the tag will be rounded according to the aggregation period parameter.

Value

Returns new myClim object in Agg-format see myClim-package When fun=NULL, period=NULL records are not modified but only converted to Agg-format. When fun and period are provided then time step is aggregated based on function.

Examples

```
hour_data <- mc_agg(mc_data_example_clean, c("min", "max", "percentile"),
                              "hour", percentiles = 50, min_coverage=0.5)
day_data <- mc_agg(mc_data_example_clean, list(TMS_T1=c("max", "min"), TMS_T2="mean"),
                             "day", min_coverage=1)
month_data <- mc_agg(mc_data_example_clean, fun=list(TMS_T3="below5"),period = "month",
                         custom_functions = list(below5=function(x){length(x[x<(-5)])}))
```

`mc_calc_cumsum` *Cumulative sum*

Description

This function creates a new virtual sensor on locality within the myClim data object. The virtual sensor represents the cumulative sum of the values on the input sensor. Names of new sensors are original sensor name + `outpus_suffix`.

Usage

```
mc_calc_cumsum(data, sensors, output_suffix = "_cumsum", localities = NULL)
```

Arguments

`data`	cleaned myClim object see myClim-package
`sensors`	names of sensors on which to calculate cumulative sum
`output_suffix`	name suffix for virtual sensor names (default "_cumsum") e.g. TMS_T3_cumsum
`localities`	list of locality_ids for calculation; if NULL then all (default NULL)

Details

If value type of sensor is logical, then output type is integer. (TRUE, TRUE, FALSE -> 2)

Value

The same myClim object as input but with added cumsum sensors.

Examples

```
cumsum_data <- mc_calc_cumsum(mc_data_example_agg, c("TMS_T1", "TMS_T2"))
```

`mc_calc_fdd` *Freezing Degree Days*

Description

This function creates a new virtual sensor on locality within the myClim data object. The new virtual sensor provides FDD Freezing Degree Days.

Usage

```
mc_calc_fdd(data, sensor, output_prefix = "FDD", t_base = 0, localities = NULL)
```

Arguments

`data`	cleaned myClim object see myClim-package
`sensor`	name of temperature sensor used for FDD calculation e.g. TMS_T3 see `names(mc_data_sensors)`
`output_prefix`	name prefix of new FDD sensor (default "FDD") name of output sensor consists of output_prefix and value t_base (FDD0_TMS_T3)
`t_base`	threshold temperature for FDD calculation (default 0)
`localities`	list of locality_ids for calculation; if NULL then all (default NULL)

Details

The allowed step length for FDD calculation is day and shorter. Function creates a new virtual sensor with the same time step as input data. For shorter time steps than the day (which is however not intuitive for FDD) the FDD value is the contribution of the time step to the freezing degree day. Be careful while aggregating freezing degree days to longer periods only meaningful aggregation function is `sum`, but myClim allows you to apply anything see `mc_agg()`.

Note that FDD is always positive number, despite summing freezing events. When you set `t_base=-1` you get the sum of degree days below -1 °C but expressed in positive number if you set `t_base=1` you get also positive number. Therefore pay attention to name of output variable which contains `t_base` value. FDD1_TMS_T3, t_base=1 vs FDDminus1_TMS_T3, t_base=-1

Value

The same myClim object as input but with added virtual FDD sensor

Examples

```
fdd_data <- mc_calc_fdd(mc_data_example_agg, "TMS_T3", localities = c("A2E32", "A6W79"))
fdd_agg <- mc_agg(fdd_data, list(TMS_T3=c("min", "max"), FDD5="sum"), period="day")
```

`mc_calc_gdd` *Growing Degree Days*

Description

This function creates a new virtual sensor for each locality within myClim data object. The new virtual sensor provides values of GDD (Growing Degree Days) in degees Celsius for each time step in the original timeseries.

Usage

```
mc_calc_gdd(data, sensor, output_prefix = "GDD", t_base = 5, localities = NULL)
```

Arguments

`data`	cleaned myClim object see myClim-package
`sensor`	name of temperature sensor used for GDD calculation e.g. TMS_T3 see `names(mc_data_sensors)`
`output_prefix`	name prefix of new GDD sensor (default "GDD" -> "GDD5_TMS_T3") name of output sensor consists of output_prefix and value t_base e.g. GDD5
`t_base`	base temperature for calculation of GDD (default 5°C)
`localities`	list of locality_ids for calculation; if NULL then all (default NULL)

Details

Function calculates growing degree days as follows: GDD = max(0;(T - Tbase)) . period(days) The maximum allowed time step length for GDD calculation is one day. Function creates a new virtual sensor with the same time step as input data. For shorter time steps than one day, the GDD value is the contribution of the interval to the growing degree day, assuming constant temperature over this period. Be careful while aggregating growing degree days to longer periods, because only meaningful aggregation function here is `sum`, but myClim let you apply any aggregation function see `mc_agg()`.

Value

The same myClim object as input but with added virtual GDD sensor

Examples

```
gdd_data <- mc_calc_gdd(mc_data_example_agg, "TMS_T3", localities = c("A2E32", "A6W79"))
gdd_agg <- mc_agg(gdd_data, list(TMS_T3=c("min", "max"), GDD5="sum"), period="day")
```

`mc_calc_snow` *Snow detection from temperature*

Description

This function creates a new virtual sensor on locality within the myClim data object. Virtual sensor hosts values of snow cover presence/absence detected from temperature time-series.

Usage

```
mc_calc_snow(
  data,
  sensor,
  output_sensor = "snow",
  localities = NULL,
  range = 1,
  tmax = 1.25,
  days = 3
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`sensor`	name of temperature sensor used for snow estimation. (e.g. TMS_T2)
`output_sensor`	name of output snow sensor (default "snow")
`localities`	list of locality_ids where snow will be calculated; if NULL then all (default NULL)
`range`	maximum temperature range threshold for snow-covered sensor (default 1°C)

tmax	maximum temperature threshold for snow-covered sensor (default 1.25°C)
days	number of days to be used for moving-window for snow detection algorithm (default 3 days)

Details

Function detects snow cover from temperature time-series. Temperature sensor is considered as covered by snow when the maximal temperature in the preceding or subsequent time-window (specified by days param) does not exceed specific tmax threshold value (default 1.25°C) and the temperature range remain below specified range threshold (default 1°C). This function rely on insulating effect of a of snow layer, significantly reducing diurnal temperature variation and restricting the maximal temperature near the ground close to freezing point. Temperature sensor near the ground (TMS_T2) is default choice for snow-cover detection from Tomst TMS loggers. Snow detection with default values accurately detects snow of depth > 15cm (unpublished data). For detection of thin snow, range parameter should be set to 3-4 °C. The function returns vector of snow cover (TRUE/FLASE) with same time-step as input data. To get number of days with snow cover and more snow summary characteristics use mc_calc_snow_agg after snow detection.

Value

myClim object with added virtual sensor 'snow' (logical) indicating snow presence/absence (TRUE/FALSE).

Examples

```
data <- mc_calc_snow(mc_data_example_agg, "TMS_T2", output_sensor="TMS_T2_snow",
                     localities = c("A2E32", "A6W79"))
```

---

mc_calc_snow_agg *Summary of TRUE/FALSE snow sensor*

Description

This function works with the virtual snow sensor of TRUE/FALSE which is the output of `mc_calc_snow()`. So, before calling mc_calc_snow_agg you need to calculate or import mc_read_ TRUE/FALSE snow sensor. mc_calc_snow_agg returns the summary table of snow sensor (e.g number of days with snow cover, first and last date of continual snow cover longer than input period). The snow summary is returned for whole date range provided. And is returned as new data.frame in contrast with other mc_calc functions returning virtual sensors.

Usage

```
mc_calc_snow_agg(
  data,
  snow_sensor = "snow",
  localities = NULL,
  period = 3,
  use_utc = FALSE
)
```

Arguments

`data`	cleaned myClim object see myClim-package with TRUE/FALSE snow sensor see `mc_calc_snow()`
`snow_sensor`	name of snow sensor containing TRUE/FALS snow detection, suitable for virtual sensors created by function `mc_calc_snow`; (default "snow")
`localities`	optional subset of localities where to run the function (list of locality_ids); if NULL then return all localities (default NULL)
`period`	number of days defining the continual snow cover period of interest (default 3 days)
`use_utc`	if set FALSE then time is shifted based on offset provided in locality metadata `tz_offset`, see e.g. `mc_prep_solar_tz()`, `mc_prep_meta_locality()`; (default FALSE)

Details

Primary designed for virtual snow sensor calculated by `mc_calc_snow()`, but accepts any sensor with TRUE/FLAST snow event detection. If `snow_sensor` on the locality is missing, then locality is skipped.

Value

Returns data.frame with columns:

- locality - locality id
- snow_days - number of days with snow cover
- first_day - first day with snow
- last_day - last day with snow
- first_day_period - first day of period with continual snow cover based on `period` parameter
- last_day_period - last day of period with continual snow cover based on `period` parameter

Examples

```
data <- mc_calc_snow(mc_data_example_agg, "TMS_T2", output_sensor="TMS_T2_snow",
                     localities = c("A2E32", "A6W79"))
mc_calc_snow_agg(data, "TMS_T2_snow")
```

`mc_calc_tomst_dendro` *Converting Tomst dendrometer values to micrometers*

Description

This function creates a new virtual sensor on locality within the myClim data object. The virtual sensor provides the values of the change in stem size converted from raw Tomst units to micrometers. Note that newer versions of Tomst Lolly software can directly convert raw Tomst units to micrometers.

Usage

```
mc_calc_tomst_dendro(
  data,
  dendro_sensor = mc_const_SENSOR_Dendro_raw,
  output_sensor = mc_const_SENSOR_dendro_l_um,
  localities = NULL
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`dendro_sensor`	name of change in stem size sensor to be converted from raw to micrometers (default "Dendro_raw") see `names(mc_data_sensors)`
`output_sensor`	name of new change in stem size sensor (default "dendro_l_um")
`localities`	list of locality_ids for calculation; if NULL then all (default NULL)

Value

myClim object same as input but with added dendro_l_um sensor

Examples

```
agg_data <- mc_calc_tomst_dendro(mc_data_example_agg, localities="A1E05")
```

`mc_calc_vpd` *Calculate vapor pressure deficit (in kPa)*

Description

This function creates a new virtual sensor on locality within the myClim data object. The virtual sensor represents the vapor pressure deficit (in kPa) calculated from temperature and relative air humidity.

Usage

```
mc_calc_vpd(
  data,
  temp_sensor = "HOBO_T",
  rh_sensor = "HOBO_RH",
  output_sensor = "VPD",
  elevation = 0,
  metadata_elevation = TRUE,
  localities = NULL
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`temp_sensor`	name of temperature sensor. Temperature sensor must be in T_C physical.
`rh_sensor`	name of relative air humidity sensor. Humidity sensor must be in RH physical.
`output_sensor`	name of new virtual VPD sensor (default "VPD")
`elevation`	value in meters (default 0)
`metadata_elevation`	if TRUE then elevation from metadata of locality is used (default TRUE)
`localities`	list of locality_ids for calculation; if NULL then all (default NULL)

Details

Equation are from the CR-5 Users Manual 2009–12 from Buck Research. These equations have been modified from Buck (1981) and adapted by Jones, 2013 (eq. 5.15) Elevation to pressure conversion function uses eq. 3.7 from Campbell G.S. & Norman J.M. (1998).

Value

myClim object same as input but with added VPD sensor

References

Jones H.G. (2014) Plants and Microclimate, Third Edit. Cambridge University Press, Cambridge Buck A.L. (1981) New equations for computing vapor pressure and enhancment factor. Journal of Applied Meteorology 20: 1527–1532. Campbell G.S. & Norman J.M. (1998). An Introduction to Environmental Biophysics, Springer New York, New York, NY

Examples

```
agg_data <- mc_calc_vpd(mc_data_example_agg, "HOBO_T", "HOBO_RH", localities="A2E32")
```

---

mc_calc_vwc — *Conversion of raw TMS soil moisture values to volumetric water content (VWC)*

Description

This function creates a new virtual sensor on the locality within the myClim data object. Function converts the raw TMS soil moisture (scaled TDT signal) to volumetric water content (VWC).

Usage

```
mc_calc_vwc(
  data,
  moist_sensor = mc_const_SENSOR_TMS_moist,
  temp_sensor = mc_const_SENSOR_TMS_T1,
  output_sensor = "VWC_moisture",
  soiltype = "universal",
  localities = NULL,
  ref_t = mc_const_CALIB_MOIST_REF_T,
  acor_t = mc_const_CALIB_MOIST_ACOR_T,
  wcor_t = mc_const_CALIB_MOIST_WCOR_T,
  frozen2NA = TRUE
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`moist_sensor`	name of soil moisture sensor to be converted from TMS moisture values to volumetric water content (default "TMS_moist") see `names(mc_data_sensors)`. Soil moisture sensor must be in moisture_raw physical units see `names(mc_data_physical)`.
`temp_sensor`	name of soil temperature sensor (default "TMS_T1") see `names(mc_data_sensors)`. Temperature sensor must be in T_C physical units.
`output_sensor`	name of new virtual sensor with VWC values (default "VWC_moisture")
`soiltype`	Either character corresponding to one of `soiltype` from mc_data_vwc_parameters (default `"universal"`), or a list with parameters `a`, `b` and `c` provided by the user as a `list(a=Value_1, b=Value_2, c=Value_3)`.
`localities`	list of locality_ids used for calculation; if NULL then all localities are used (default NULL)
`ref_t`	(default 24)
`acor_t`	(default 1.91132689118083) correction parameter for temperature drift in the air, see `mc_calib_moisture()`
`wcor_t`	(default 0.64108) correction parameter for temperature drift in the water, see `mc_calib_moisture()`
`frozen2NA`	if TRUE then VWC values are set to NA when the soil temperature is below 0 °C (default TRUE)

Details

This function is suitable for TOMST TMS loggers measuring soil moisture in raw TMS units. The raw TMS units represents inverted and numerically rescaled (1-4095) electromagnetic signal from the moisture sensor working on Time Domain Transmission principle (Wild et al. 2019). For TMS4 logger, the typical raw TMS moisture values range from cca 115 units in dry air to cca 3635 units in distilled water - see mc_calib_moisture.

Raw TMS moisture values can be converted to the soil volumetric water content with calibration curves. The function provides several experimentally derived calibration curves which were developped at reference temperature. To account for the difference between reference and actual temperature, the function uses actual soil temperature values measured by TMS_T1 soil temperature sensor.

The default calibration curve is "universal", which was designed for mineral soils (see Kopecký et al. 2021). Specific calibration curves were developed for several soil types (see Wild et al. 2019) and the user can choose one of these or can define its own calibration - see mc_data_vwc_parameters

Currently available calibration curves are: sand, loamy sand A, loamy sand B, sandy loam A, sandy loam B, loam, silt loam, peat, water, universal, sand TMS1, loamy sand TMS1, silt loam TMS1. For details see mc_data_vwc_parameters.

It is also possible to define a new calibarion function with custom parameters `a`, `b` and `c`. These can be derived e.g. from TOMST TMS Calibr utility after entering custom ratio of clay, silt, sand.

Warning: TOMST TMS Calibr utility was developed for TMS3 series of TMS loggers, which have different range of raw soil moisture values than TMS4 series.

The function by default replace the moisture records in frozen soils with NA (param *frozen2NA*), because the TMS soil moisture sensor was not designed to measure in frozen soils and the returned values are thus not comparable with values from non-frozen soil.

Value

myClim object same as input but with added virtual VWC moisture sensor

References

Wild, J., Kopecký, M., Macek, M., Šanda, M., Jankovec, J., Haase, T. (2019) Climate at ecologically relevant scales: A new temperature and soil moisture logger for long-term microclimate measurement. Agriculture and Forest Meteorology 268, 40-47. https://doi.org/10.1016/j.agrformet.2018.12.018

Kopecký, M., Macek, M., Wild, J. (2021) Topographic Wetness Index calculation guidelines based on measured soil moisture and plant species composition. Science of the Total Environment 757, 143785. https://doi.org/10.1016/j.scitotenv.2020.143785

See Also

mc_data_vwc_parameters

Examples

```
data1 <- mc_calc_vwc(mc_data_example_agg, soiltype="sand", localities="A2E32")
data2 <- mc_calc_vwc(mc_data_example_agg, localities="A2E32",
                     soiltype=list(a=-3.00e-09, b=0.000161192, c=-0.109956505))
```

mc_calib_moisture — *Calculates coefficients for TMS moisture conversion to VWC*

Description

Specialized function for calibration of TOMST TMS moisture sensor. Function calculate correction parameters for individual logger (slope and intercept) from TMS moisture measurements in demineralized water and dry air.

Usage

```
mc_calib_moisture(
  raw_air,
  raw_water,
  t_air = 24,
  t_water = 24,
  ref_air = 114.534,
  ref_water = 3634.723,
  ref_t = mc_const_CALIB_MOIST_REF_T,
  acor_t = mc_const_CALIB_MOIST_ACOR_T,
  wcor_t = mc_const_CALIB_MOIST_WCOR_T
)
```

Arguments

`raw_air`	Raw TMS moisture signal in air
`raw_water`	Raw TMS moisture signal in water
`t_air`	temperature of air (default 24)
`t_water`	temperature of water (default 24)
`ref_air`	raw air signal of reference logger used to derive soil calibration parameters (default 114.534)
`ref_water`	raw air signal of reference logger used to derive soil calibration parameters (default 3634.723)
`ref_t`	reference logger temperature (default 24)
`acor_t`	temperature drift correction parameter in the air (default 1.911)
`wcor_t`	temperature drift correction parameter in the water (default 0.641)

Details

This function calculate calibration parameters `cor_factor` and `cor_intercept` accounting for individual differencies in TMS moisture sensor signal in air and in water against reference loggers which were used for estimation of parameters of soil VWC conversion curves. These parameters must be loaded into myClim object `mc_prep_calib_load()` prior to calling `mc_calc_vwc()`. Parameters for soils available in my_Clim were derived for TMS3 logger version, with slightly different typical air and water signal. Correction parameters for TMS4 loggers therefore can be expected in the range of values: cor_factor = (-150; -450) and cor_slope = (100, 450)

Value

list with correction factor and correction slope

Examples

```
# load example data
files <- c(system.file("extdata", "data_94184102_0.csv", package = "myClim"))
tomst_data <- mc_read_files(files, "TOMST")

# vwc without calibration
tomst_data <- mc_calc_vwc(tomst_data, soiltype = "universal", output_sensor = "VWC_universal")

# load calibration
my_cor <- mc_calib_moisture(raw_air = 394, raw_water = 3728, t_air = 21, t_water = 20)
my_calib_tb <- data.frame(serial_number = c("94184102"), sensor_id = "TMS_moist",
                          datetime = as.POSIXct("2020-01-01 00:00"),
                          cor_factor = my_cor$cor_factor, cor_slope = my_cor$cor_slope)
tomst_data_cal <- mc_prep_calib_load(tomst_data, my_calib_tb)
# vwc using calibration
tomst_data_cal <- mc_calc_vwc(tomst_data_cal, soiltype = "universal",
                              output_sensor = "VWC_universal_calib")
# plot results
## Not run:
sensors <- mc_info(tomst_data_cal)$sensor_name
mc_plot_line(tomst_data_cal, sensors = c(sensors[startsWith(sensors,"VWC")])
     + ggplot2::scale_color_viridis_d(begin = 0.2, end = 0.8))
## End(Not run)
```

---

mc_const_CALIB_MOIST_ACOR_T

Default temperature drift for TMS moisture in the air.

Description

1.91132689118083 = default temperature drift correction parameter in the air - TMS moisture sensor. This constant is used in the function mc_calc_vwc.

Usage

```
mc_const_CALIB_MOIST_ACOR_T
```

Format

An object of class `numeric` of length 1.

mc_const_CALIB_MOIST_REF_T
Default ref. temperate for TMS moisture calibration

Description

24°C = default reference calibration temperate for TMS moisture sensor

Usage

```
mc_const_CALIB_MOIST_REF_T
```

Format

An object of class `numeric` of length 1.

mc_const_CALIB_MOIST_WCOR_T
Default temperature drift for TMS moisture in the water

Description

0.64108 = default temperature drift correction parameter in the water - TMS moisture sensor. This constant is used in the function mc_calc_vwc.

Usage

```
mc_const_CALIB_MOIST_WCOR_T
```

Format

An object of class `numeric` of length 1.

mc_const_SENSOR_count
Count sensor id see `mc_agg()`

Description

Count sensor id see `mc_agg()`

Usage

```
mc_const_SENSOR_count
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_coverage

Coverage sensor id see `mc_agg()`

Description

Coverage sensor id see `mc_agg()`

Usage

```
mc_const_SENSOR_coverage
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_dendro_l_um

Radius difference sensor id

Description

Radius difference sensor id

Usage

```
mc_const_SENSOR_dendro_l_um
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_Dendro_raw`
Default sensor for TOMST Dendrometer radius difference

Description

This constant is used in the function mc_calc_tomst_dendro as default sensor for converting the change in stem size from raw TOMST units to micrometers. mc_const_SENSOR_Dendro_raw = "Dendro_raw"

Usage

```
mc_const_SENSOR_Dendro_raw
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_Dendro_T`
Default sensor for TOMST Dendrometer temperature

Description

Default sensor for TOMST Dendrometer temperature

Usage

```
mc_const_SENSOR_Dendro_T
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_FDD *Freezing Degree Days sensor id see* mc_calc_fdd()

Description

Freezing Degree Days sensor id see mc_calc_fdd()

Usage

```
mc_const_SENSOR_FDD
```

Format

An object of class character of length 1.

mc_const_SENSOR_GDD *Growing Degree Days sensor id see* mc_calc_gdd()

Description

Growing Degree Days sensor id see mc_calc_gdd()

Usage

```
mc_const_SENSOR_GDD
```

Format

An object of class character of length 1.

mc_const_SENSOR_HOBO_EXTT *Onset HOBO external temperature sensor id*

Description

Onset HOBO external temperature sensor id

Usage

```
mc_const_SENSOR_HOBO_EXTT
```

Format

An object of class character of length 1.

`mc_const_SENSOR_HOBO_RH` *Onset HOBO humidity sensor id*

Description

Onset HOBO humidity sensor id

Usage

```
mc_const_SENSOR_HOBO_RH
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_HOBO_T` *Onset HOBO temperature sensor id*

Description

Onset HOBO temperature sensor id

Usage

```
mc_const_SENSOR_HOBO_T
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_integer` *General integer sensor id*

Description

General integer sensor id

Usage

```
mc_const_SENSOR_integer
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_logical *General logical sensor id*

Description

General logical sensor id

Usage

```
mc_const_SENSOR_logical
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_precipitation *Precipitation sensor id*

Description

Precipitation sensor id

Usage

```
mc_const_SENSOR_precipitation
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_real *General real sensor id*

Description

General real sensor id

Usage

```
mc_const_SENSOR_real
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_RH *Relative humidity sensor id*

Description

Relative humidity sensor id

Usage

```
mc_const_SENSOR_RH
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_snow_bool *Snow existence sensor id see mc_calc_snow()*

Description

Snow existence sensor id see `mc_calc_snow()`

Usage

```
mc_const_SENSOR_snow_bool
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_snow_fresh *Height of newly fallen snow sensor id*

Description

Height of newly fallen snow sensor id

Usage

```
mc_const_SENSOR_snow_fresh
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_snow_total`
Height snow sensor id

Description

Height snow sensor id

Usage

```
mc_const_SENSOR_snow_total
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_sun_shine`
Time of sun shine sensor id

Description

Time of sun shine sensor id

Usage

```
mc_const_SENSOR_sun_shine
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_Thermo_T`
Default sensor for TOMST Thermologger temperature

Description

Default sensor for TOMST Thermologger temperature

Usage

```
mc_const_SENSOR_Thermo_T
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_TMS_moist

Default sensor for TOMST TMS raw soil moisture

Description

This constant is used in the function mc_calc_vwc as default for sensor for converting the raw TMS soil moisture (scaled TDT signal) to volumetric water content (VWC). mc_const_SENSOR_TMS_moist = "TMS_moist"

Usage

```
mc_const_SENSOR_TMS_moist
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_TMS_T1

Default sensor for TOMST TMS soil temperature

Description

This constant is used in the function mc_calc_vwc to account for soil temperature effect while converting the raw TMS soil moisture (scaled TDT signal) to volumetric water content (VWC). mc_const_SENSOR_TMS_T1 = "TMS_T1"

Usage

```
mc_const_SENSOR_TMS_T1
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_TMS_T2 *Default sensor for TOMST TMS temperature of soil surface*

Description

Default sensor for TOMST TMS temperature of soil surface

Usage

```
mc_const_SENSOR_TMS_T2
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_TMS_T3 *Default sensor for TOMST TMS air temperature*

Description

Default sensor for TOMST TMS air temperature

Usage

```
mc_const_SENSOR_TMS_T3
```

Format

An object of class `character` of length 1.

mc_const_SENSOR_T_C *Temperature sensor id*

Description

Temperature sensor id

Usage

```
mc_const_SENSOR_T_C
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_VPD` *Vapor Pressure Deficit sensor id see* `mc_calc_vpd()`

Description

Vapor Pressure Deficit sensor id see `mc_calc_vpd()`

Usage

```
mc_const_SENSOR_VPD
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_VWC` *Volumetric soil moisture sensor id see* `mc_calc_vwc()`

Description

Volumetric soil moisture sensor id see `mc_calc_vwc()`

Usage

```
mc_const_SENSOR_VWC
```

Format

An object of class `character` of length 1.

`mc_const_SENSOR_wind_speed` *Speed of wind sensor id*

Description

Speed of wind sensor id

Usage

```
mc_const_SENSOR_wind_speed
```

Format

An object of class `character` of length 1.

`mc_DataFormat-class` *Class for Logger File Data Format*

Description

This class is used for parsing source TXT/CSV files downloaded from microclimatic loggers.

Details

myClim offers several pre-defined logger file data formats, such as TOMST TMS or HOBO. Users can also define custom readings for their own loggers. Pre-defined and custom loggers in myClim each have their own specific object of class `mc_{logger}DataFormat`, which defines the parameters for handling logger files. The pre-defined logger definitions are stored in the R environment object `./data/mc_data_formats.rda`.

Slots

`skip` The number of rows to skip before the first row containing microclimatic records. For example, to skip the header (default 0).

`separator` The column separator (default is a comma ",").

`date_column` The index of the date column - required (default NA).

`date_format` The format of the date (default NA).

For a description of the date_format parameter, see `strptime()`. If the format is in ISO8601 and the function `vroom::vroom()` automatically detects datetime values, the date_format parameter can be NA.

`na_strings` Strings for representing NA values, e.g., "-100", "9999" (default "").

`error_value` The value that represents an error of the sensor, e.g., 404, 9999 (default NA).

The error_value is replaced by NA, and intervals of errors are flagged in `sensor$states` (see myClim-package).

`columns` A list with names and indexes of value columns - required (default list()).

Names come from names(mc_data_sensors). Names are defined as constants `mc_const_SENSOR_*`. For example, if the third column is temperature, you can define it as `columns[[mc_const_SENSOR_T_C]] <- 3`. There are universal sensors for arbitrary value types: `mc_const_SENSOR_real`, `mc_const_SENSOR_integer` and `mc_const_SENSOR_logical`. Multiple columns with same sensor type can be defined as `columns[[mc_const_SENSOR_real]] <- c(2, 3, 4)`. The names in this example will be `real1`, `real2` and `real3`.

`col_types` Parameter for `vroom::vroom()` (default NA).

To ensure the correct reading of values, you have the possibility to strictly define the types of columns.

`filename_serial_number_pattern` A character pattern for detecting the serial number from the file name (default NA).

The regular expression with brackets around the serial number. For example, the pattern for old TOMST files is `"data_(\\d+)_\\d+\\.csv$"`. If the value is NA, the name of the file is used as the serial number.

`data_row_pattern` A character pattern for detecting the correct file format (default NA).

The regular expression. If data_row_pattern is NA, then the file format is not checked.

`logger_type` The type of logger: TMS, TMS_L45, Thermo, Dendro, HOBO, ... (default NA).

`tz_offset` The timezone offset in minutes from UTC - required (default NA).

If the value of the tz_offset parameter is 0, then datetime values are in UTC. If the time zone offset is defined in the value, e.g., `"2020-10-06 09:00:00+0100"`, and `date_format` is `"%Y-%m-%d %H:%M:%S%z"`, the value is automatically converted to UTC.

See Also

mc_data_formats, mc_TOMSTDataFormat, mc_TOMSTJoinDataFormat, mc_HOBODataFormat

---

`mc_data_example_agg` *Example data in Agg-format.*

Description

Cleaned data in Agg-format. Three example localities situated in Saxon Switzerland National Park. myClim object has metadata and covers time period from 2020-10 to 2021-02.

Data includes time-series from 4 loggers:

- Tomst TMS4 with 4 sensors ("TMS_T1", "TMS_T2", "TMS_T3", "TMS_moist")
- Tomst Thermologger with 1 sensor ("Thermo_T)
- Tomst Point Dendrometer with 2 sensors ("Dendro_T", "Dendro_raw")
- HOBO U23 with 2 sensors ("HOBO_T", "HOBO_RH")

Usage

```
mc_data_example_agg
```

Format

An object of class `myClimList` (inherits from `list`) of length 2.

`mc_data_example_clean` *Example cleaned data in Raw-format.*

Description

Cleaned data. Three example localities situated in Saxon Switzerland National Park. myClim object has metadata and covers time period from 2020-10 to 2021-02.

Data includes time-series from 4 loggers:

- Tomst TMS4 with 4 sensors ("TMS_T1", "TMS_T2", "TMS_T3", "TMS_moist")
- Tomst Thermologger with 1 sensor ("Thermo_T)
- Tomst Point Dendrometer with 2 sensors ("Dendro_T", "Dendro_raw")
- HOBO U23 with 2 sensors ("HOBO_T", "HOBO_RH")

Usage

```
mc_data_example_clean
```

Format

An object of class `myClimList` (inherits from `list`) of length 2.

`mc_data_example_raw` *Example data in Raw-format*

Description

Raw data, not cleaned. Three example localities situated in Saxon Switzerland National Park. myClim object has metadata and covers time period from 2020-10 to 2021-02.

Data includes time-series from 4 loggers:

- Tomst TMS4 with 4 sensors ("TMS_T1", "TMS_T2", "TMS_T3", "TMS_moist")
- Tomst Thermologger with 1 sensor ("Thermo_T)
- Tomst Point Dendrometer with 2 sensors ("Dendro_T", "Dendro_raw")
- HOBO U23 with 2 sensors ("HOBO_T", "HOBO_RH")

Usage

```
mc_data_example_raw
```

Format

An object of class `myClimList` (inherits from `list`) of length 2.

`mc_data_formats` *Formats of source data files*

Description

R object of class environment with the definitions how to parse specific microclimatic logger files. In case you would like to add new, unsupported logger, this is the place where the reading key is stored.

Usage

```
mc_data_formats
```

Format

An object of class `environment` of length 3.

Details

Package myClim support formats TOMST, TOMST_join and HOBO. The environment object is stored in `./data/mc_data_formats.rda`.

TOMST

TOMST data format has defined structure. Expected name of data file is in format data_\<serial_number\>_\<x\>.csv. Value serial_number can be automatically detected from file name. Datetime is in UTC and is stored in col 2. Temperature values are stored in col 3-5. Moisture () Supported logger types are TMS (for TMS-3/TMS-4), ThermoDataLogger (for Thermologger), Dendrometer and TMS_L45 (for TMS-4 Long 45cm).

TOMST_join

TOMST_join data format is used by output files from JoinTMS.exe software and from tupomanager.exe (TMS-1). Datetime in col 4, temperatures in col 5-7, moisture in col 8.

HOBO

HOBO data format is export format from software HOBOware of Onset company for HOBO U23 Pro v2 loggers (Temperature/RH). Format is very variable and can be adjusted by user in preferences of HOBOware. Strucuture of HOBO files format can be partly detected automatically from header of data. Format of date-time (`date_format`) must be set manually in myClim reading functions (`mc_read_files()`, `mc_read_data()`). Date and time separated in more columns is not supported in myClim reading. If time zone is not defined in header of HOBO txt or csv file and is not UTC, then `tz_offset` must be filled in while reading. UTF-8 encoding of HOBO file is required for reding to myClim.

See Also

mc_DataFormat, mc_TOMSTDataFormat, mc_TOMSTJoinDataFormat, mc_HOBODataFormat `mc_data_heights` *Default heights of sensors*

Description

This table is used to set the default heights in metadata of sensors based on logger type. The defaults were set based on the most common uses, defaults can be overwrite be user. see mc_prep_meta_sensor

Usage

```
mc_data_heights
```

Format

An object of class `data.frame` with 15 rows and 4 columns.

Details

data.frame with columns:

- logger_type
- sensor_name
- height - character representation of height
- suffix - suffix for sensor_name. If suffix is NA, then sensor_name is not modified.

Default heights are:

TOMST - Thermo

- Thermo_T = air 200 cm

TOMST - TMS

- TMS_T1 = soil 8 cm
- TMS_T2 = air 2 cm
- TMS_T3 = air 15 cm
- TMS_moist = soil 0-15 cm

TOMST - Dendro

- Dendro_T = 130 cm
- Dendro_raw = 130 cm

TOMST - TMS_L45

- TMS_T1 = soil 40 cm
- TMS_T2 = soil 30 cm
- TMS_T3 = air 15 cm

- TMS_moist = soil 30-44 cm

HOBO - HOBO_U23-001A

- HOBO_T = air 150 cm
- HOBO_RH = air 150 cm

HOBO - HOBO_U23-004

- HOBO_T = air 2 cm
- HOBO_extT = soil 8 cm

See Also

`mc_read_files()`, `mc_read_data()`

`mc_data_physical` *Physical quantities definition*

Description

R object of class environment with the definitions of physical elements for recording the microclimate e.g. temperature, speed, depth, volumetric water content... see mc_Physical. Similarly as in case of logger format definitions mc_DataFormat it is easy to add new, physical here.

Usage

```
mc_data_physical
```

Format

An object of class `environment` of length 11.

See Also

mc_Physical

Currently supported physical elements:

- l_cm - length in cm
- l_mm - length in mm
- l_um - length in um
- VWC - volumetric moisture in m3/m3
- RH - relative humidity in %
- T_C - temperature in °C
- t_h - time in hours
- moisture_raw - raw TMS moisture sensor values
- radius_raw - radius difference in raw units
- v - speed in m/s

mc_data_sensors *Sensors definition.*

Description

R object of class environment with the definitions of (micro)climatic sensors. see mc_Sensor. Similarly as in case of logger format definitions mc_DataFormat it is easy to add new, sensor here. There is also universal sensor `real` where you can store any real values.

Usage

```
mc_data_sensors
```

Format

An object of class `environment` of length 28.

Details

Names of items are sensor_ids. Currently supported sensors:

- count - result of `count` function `mc_agg()`
- coverage - result of `coverage` function `mc_agg()`
- Dendro_T - temperature in Tomst dendrometer (°C)
- Dendro_raw - change in stem size in Tomst dendrometer (raw units) `mc_calc_tomst_dendro()`
- dendro_l_um - change in stem size (um) `mc_calc_tomst_dendro()`
- FDD - result of function `mc_calc_fdd()`
- GDD - result of function `mc_calc_gdd()`
- HOBO_RH - relative humidity in HOBO U23-001A logger (%)
- HOBO_T - temperature in HOBO U23 logger (°C)
- HOBO_extT - external temperature in HOBO U23-004 logger (°C)
- integer - universal sensor with integer values
- logical - universal sensor with logical values
- VWC - volumetric water content in soil (m3/m3)
- precipitation - (mm)
- real - universal sensor with real values
- RH - relative humidity sensor (%)
- snow_bool - result of function `mc_calc_snow()`
- snow_fresh - fresh snow height (cm)
- snow_total - total snow height (cm)
- sun_shine - time of sun shine (hours)

- T_C - universal temperature sensor (°C)
- Thermo_T - temperature sensor in Tomst Thermologger (°C)
- TMS_T1 - soil temperature sensor in Tomst TMS (°C)
- TMS_T2 - surface temperature sensor in Tomst TMS (°C)
- TMS_T3 - air temperature sensor in Tomst TMS (°C)
- TMS_moist - soil moisture sensor in Tomst TMS (raw TMS units)
- wind - wind speed (m/s)

`mc_data_vwc_parameters`

Volumetric water content parameters

Description

Data frame hosting the coefficients for the conversion of TMS raw moisture units to volumetric warer content. The coefficients come from laboratory calibration for several soil types. For the best performance you should specify the soil type in case you know it and in case it could be approximated to the available calibration e.g sand, loam, loamy sand.... See `mc_calc_vwc()`

Usage

```
mc_data_vwc_parameters
```

Format

An object of class `data.frame` with 13 rows and 9 columns.

Details

data.frame with columns:

- soiltype
- a
- b
- c
- rho
- clay
- silt
- sand
- ref

References

Wild, J., Kopecky, M., Macek, M., Sanda, M., Jankovec, J., Haase, T., 2019. Climate at ecologically relevant scales: A new temperature and soil moisture logger for long-term microclimate measurement. Agric. For. Meteorol. 268, 40-47. https://doi.org/10.1016/j.agrformet.2018.12.018

Kopecky, M., Macek, M., Wild, J., 2021. Topographic Wetness Index calculation guidelines based on measured soil moisture and plant species composition. Sci. Total Environ. 757, 143785. https://doi.org/10.1016/j.scitotenv.2020.143785

---

mc_env_moist *Standardised myClim soil moisture variables*

Description

The wrapper function returning 4 standardised and ecologically relevant myClim variables derived from soil moisture measurements. The mc_env_moist function needs time-series of volumetric water content (VWC) measurements as input. Therefore, non-VWC soil moisture measurements must be first converted to VWC. For TMS loggers see `mc_calc_vwc()`

Usage

```
mc_env_moist(
  data,
  period,
  use_utc = TRUE,
  custom_start = NULL,
  custom_end = NULL,
  min_coverage = 1
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`period`	output period see `mc_agg()`
`use_utc`	if FALSE, then local time is used for day aggregation see `mc_agg()` (default TRUE)
`custom_start`	start date for custom period see `mc_agg()` (default NULL)
`custom_end`	end date for custom period see `mc_agg()` (default NULL)
`min_coverage`	the threshold specifying how many missing values can you accept within aggregation period. see `mc_agg()` value from range 0-1 (default 1)

Details

This function was designed for time-series of step shorter than one day and will not work with coarser data. In contrast with other myClim functions returning myClim objects, this wrapper function returns long table. Variables are named based on sensor name, height, and function e.g., (VWC.soil_0_15_cm.5p, VWC.soil_0_15_cm.mean)

Standardised myClim soil moisture variables:

- VWC.5p: Minimum soil moisture = 5th percentile of VWC values
- VWC.mean: Mean soil moisture = mean of VWC values
- VWC.95p: Maximum soil moisture = 95th percentile of VWC values
- VWC.sd: Standard deviation of VWC measurements

Value

table in long format with standardised myClim variables

Examples

```
data <- mc_prep_crop(mc_data_example_agg, lubridate::ymd_h("2020-11-01 00"),
                     lubridate::ymd_h("2021-02-01 00"), end_included = FALSE)
data <- mc_calc_vwc(data, localities=c("A2E32", "A6W79"))
mc_env_moist(data, "month")
```

---

mc_env_temp *Standardised myClim temperature variables*

Description

The wrapper function returning 7 standardised and ecologically relevant myClim variables derived from temperature measurements.

Usage

```
mc_env_temp(
  data,
  period,
  use_utc = TRUE,
  custom_start = NULL,
  custom_end = NULL,
  min_coverage = 1,
  gdd_t_base = 5,
  fdd_t_base = 0
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`period`	output period see `mc_agg()`
`use_utc`	if FALSE, then local time is used for day aggregation see `mc_agg()` (default TRUE)
`custom_start`	start date for custom period see `mc_agg()` (default NULL)
`custom_end`	end date for custom period see `mc_agg()` (default NULL)
`min_coverage`	the threshold specifying how many missing values can you accept within aggregation period. see `mc_agg()` value from range 0-1 (default 1)
`gdd_t_base`	base temperature for Growing Degree Days `mc_calc_gdd()` (default 5)
`fdd_t_base`	base temperature for Freezing Degree Days `mc_calc_fdd()` (default 0)

Details

This function was designed for time-series of step shorter than one day and will not work with coarser data. It automatically use all available sensors in myClim object and returns all possible variables based on sensor type and measurement height/depth. In contrast with other myClim functions returning myClim objects, this wrapper function returns long table. The mc_env_temp function first aggregates time-series to daily time-step and then aggregates to the final time-step set in `period` parameter. Because freezing and growing degree days are always aggregated with sum function, these two variables are not first aggregated to the daily time-steps. Variables are named based on sensor name, height, and function e.g., (T.air_15_cm.max95p, T.air_15_cm.drange)

Standardised myClim temperature variables:

- min5p: Minimum temperature = 5th percentile of daily minimum temperatures
- mean: Mean temperature = mean of daily mean temperatures
- max95p: Maximum temperature = 95th percentile of daily maximum temperatures
- drange: Temperature range = mean of daily temperature range (i.e., difference between daily minima and maxima)
- GDD5: Growing degree days = sum of growing degree days above defined base temperature (default 5°C) `gdd_t_base`
- FDD0: Freezing degree days = sum of freezing degree days bellow defined base temperature (default 0°C) `fdd_t_base`
- frostdays: Frost days = number of days with frost (daily minimum < 0°C) `fdd_t_base`

Value

table in long format with standardised myClim variables

Examples

```
data <- mc_prep_crop(mc_data_example_clean, lubridate::ymd_h("2020-11-01 00"),
                     lubridate::ymd_h("2021-02-01 00"), end_included = FALSE)
mc_env_temp(data, "month")
```

mc_env_vpd — *Standardised myClim vapor pressure deficit variables*

Description

The wrapper function returning 2 standardised and ecologically relevant myClim variables derived from vapor pressure deficit. The mc_env_vpd function needs time-series of vapor pressure deficit measurements as input. Therefore, VPD must be first calculated from temperature and air humidity measurements - see `mc_calc_vpd()`

Usage

```
mc_env_vpd(
  data,
  period,
  use_utc = TRUE,
  custom_start = NULL,
  custom_end = NULL,
  min_coverage = 1
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`period`	output period see `mc_agg()`
`use_utc`	if FALSE, then local time is used for day aggregation see `mc_agg()` (default TRUE)
`custom_start`	start date for custom period see `mc_agg()` (default NULL)
`custom_end`	end date for custom period see `mc_agg()` (default NULL)
`min_coverage`	the threshold specifying how many missing values can you accept within aggregation period. see `mc_agg()` value from range 0-1 (default 1)

Details

This function was designed for time-series of step shorter than one day and will not work with coarser data. The mc_env_vpd function first aggregates time-series to daily time-step and then aggregates to the final time-step set in `period` parameter. In contrast with other myClim functions returning myClim objects, this wrapper function returns long table. Variables are named based on sensor name, height, and function e.g., (VPD.air_150_cm.mean, VPD.air_150_cm.max95p)

Standardised myClim vapor pressure deficit variables:

- VPD.mean: Mean vapor pressure deficit = mean of daily mean VPD
- VPD.max95p: Maximum vapor pressure deficit = 95th percentile of daily maximum VPD

Value

table in long format with standardised myClim variables

mc_filter — *Filter data from myClim object*

Description

This function filter data by localities, logger types and sensors.

Usage

```
mc_filter(
  data,
  localities = NULL,
  sensors = NULL,
  reverse = FALSE,
  stop_if_empty = TRUE,
  logger_types = NULL
)
```

Arguments

`data`	myClim object see myClim-package
`localities`	locality_ids for filtering data; if NULL then do nothing (default NULL)
`sensors`	sensor_names for filtering data; if NULL then do nothing see `names(mc_data_sensors)` (default NULL)
`reverse`	if TRUE then input localities and/or sensors are excluded (default FALSE)
`stop_if_empty`	if TRUE then error for empty output (default TRUE)
`logger_types`	types of logger for filtering data; if NULL then do nothing (default NULL). The logger_types parameter can by used only for raw data format see myClim-package.

Details

In default settings it returns the object containing input localities / logger types / sensors. When you provide vector of localities e.g. `localities=c("A6W79", "A2E32")` selected localities are filtered with all loggers / sensors on those localities. The same as When you provide vector of logger_types `logger_types=c("TMS", "TMS_L45")` selected loggers by type are filtered through all localities (logger_types criterion is applicable only for raw data format see myClim-package) and the sensors parameter `sensors=c("TMS_T1", "TMS_T2")`, selected sensors are filtered through all localities. When you combine localities, logger_types and sensors, then filtering return selected sensors in selected loggers on selected localities.

Parameter `reverse = TRUE` returns myClim object without listed localities, or logger types or sensors. Using `reverse = TRUE` is not allowed for combination of localities and logger types and sensors. It is allowed to use `reverse` only with single filter criterion either locality, logger type or sensor.

- reverse = TRUE and logger_types are selected then the listed logger types are removed from all localities.
- reverse = TRUE and localities are selected then the listed localities are removed from myClim object.
- reverse = TRUE and sensors are selected then listed sensors are removed from all loggers / localities.

Value

filtered myClim object

Examples

```
## keep only "A6W79", "A2E32" localities with all their sensors
filtered_data <- mc_filter(mc_data_example_raw, localities=c("A6W79", "A2E32"))

## remove "A6W79", "A2E32" localities and keep all others
filtered_data <- mc_filter(mc_data_example_raw, localities=c("A6W79", "A2E32"), reverse=TRUE)

## keep only "TMS_T1", and "TMS_T2" sensors on all localities
filtered_data <- mc_filter(mc_data_example_raw, sensors=c("TMS_T1", "TMS_T2"))

## remove "TMS_T1", and "TMS_T2" sensors from all localities
filtered_data <- mc_filter(mc_data_example_raw, sensors=c("TMS_T1", "TMS_T2"),reverse=TRUE)

## keep only "TMS_T1", and "TMS_T2" sensors on "A6W79", "A2E32" localities
filtered_data <- mc_filter(mc_data_example_raw, localities=c("A6W79", "A2E32"),
                           sensors=c("TMS_T1", "TMS_T2"))
## Remove "Dendro" loggers on all localities
filtered_data <- mc_filter(mc_data_example_raw, logger_types="Dendro", reverse=TRUE)
```

---

mc_HOBODataFormat-class

Class for reading HOBO logger files

Description

Provides the key for reading the HOBO source files. In which column is the date, in what format is the date, in which columns are records of which sensors. The code defining the class is in section methods ./R/model.R

Slots

convert_fahrenheit if TRUE temperature values are converted from °F to °C (default FALSE)

See Also

mc_DataFormat, mc_data_formats

`mc_info` *Get sensors info table*

Description

This function return data.frame with info about sensors

Usage

```
mc_info(data)
```

Arguments

data	myClim object see myClim-package

Value

data.frame with columns:

- locality_id - when provided by user then locality ID, when not provided identical with serial number
- serial_number - serial number of logger when provided or automatically detected from file name or header
- sensor_id - original sensor id (e.g.,"GDD", "HOBO_T" ,"TMS_T1", "TMS_T2")
- sensor_name - original sensor id if not modified, if renamed then new name (e.g.,"GDD5", "HOBO_T_mean" ,"TMS_T1_max", "my_sensor01")
- start_date - the oldest record on the sensor
- end_date - the newest record on the sensor
- step_seconds - time step of records series (seconds)
- period - time step of records series (text)
- min_value - minimal recorded values
- max_value - maximal recorded value
- count_values - number of non NA records
- count_na - number of NA records

Examples

```
mc_info(mc_data_example_agg)
```

mc_info_clean *Call cleaning log*

Description

This function return data.frame with information from cleaning the loggers time series see `mc_prep_clean()`

Usage

```
mc_info_clean(data)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package

Value

data.frame with columns:

- locality_id - when provided by user then locality ID, when not provided identical with serial number
- serial_number - serial number of logger when provided or automatically detected from file name or header
- start_date - date of the first record on the logger
- end_date - date of the last record on the logger
- step_seconds - detected time step in seconds of the logger measurements.
- count_duplicities - number of duplicated records (identical time)
- count_missing - number of missing records (logger outage in time when it should record)
- count_disordered - number of records incorrectly ordered in time (newer followed by older)
- rounded - T/F indication whether myClim automatically rounded time series minutes to the closes half (HH:00, HH:30) e.g. 13:07 -> 13:00

See Also

`mc_prep_clean()`

`mc_info_count` *Count data*

Description

This function return data.frame with the number of localities, loggers and sensors of input myClim object.

Usage

```
mc_info_count(data)
```

Arguments

`data`	myClim object see myClim-package

Value

data.frame with count of localities, loggers and sensors

Examples

```
count_table <- mc_info_count(mc_data_example_raw)
```

`mc_info_join` *Get joining info table*

Description

This function returns a data.frame that contains information about the join operations. Although this function performs the join process, it only returns an overview table, not the actual joined data.

Usage

```
mc_info_join(data, comp_sensors = NULL)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package
`comp_sensors`	parameter for `mc_join()` function (default NULL)

Details

This function is designed to work only with myClim objects in **Raw-format**, where the loggers are organized at localities. In **Agg-format**, myClim objects do not support loggers; sensors are directly connected to the locality. See myClim-package. `mc_info_join` does not work in Agg-format.

Value

A data.frame with the following columns:

- locality_id - The ID of the locality.
- count_loggers - Number of loggers before the join operation.
- count_joined_loggers - Number of loggers after the join operation.
- count_data_conflicts - Number of different values in overlapping sensors.
- count_errors - Number of join-related errors. An error occurs when all sensors of the loggers have different names.

`mc_info_logger` *Get loggers info table*

Description

This function returns a data.frame with information about loggers.

Usage

```
mc_info_logger(data)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package

Details

This function is designed to work only with myClim objects in **Raw-format**, where the loggers are organized at localities. In **Agg-format**, myClim objects do not support loggers; sensors are directly connected to the locality. See myClim-package. `mc_info_logger` does not work in Agg-format.

Value

A data.frame with the following columns:

- locality_id - If provided by the user, it represents the locality ID; if not provided, it is identical to the logger's serial number.
- index - Logger index at the locality.
- serial_number - Serial number of the logger, either provided by the user or automatically detected from the file name or header.
- logger_type - Logger type.
- start_date - The oldest record on the logger.
- end_date - The newest record on the logger.
- step_seconds - Time step of the record series (in seconds).

Examples

```
mc_info_logger(mc_data_example_raw)
```

mc_info_meta *Get localities metadata table*

Description

This function return data.frame with localities metadata

Usage

```
mc_info_meta(data)
```

Arguments

`data`	myClim object see myClim-package

Value

data.frame with columns:

- locality_id
- lon_wgs84
- lat_wgs84
- elevation
- tz_offset

Examples

```
mc_info_meta(mc_data_example_agg)
```

mc_info_range *Get table of sensors range*

Description

This function return data.frame with sensors range (min value, max value) and possible jumps.

Usage

```
mc_info_range(data)
```

Arguments

`data`	myClim object see myClim-package

Details

This function is mainly useful to prepare input parameter for `mc_states_outlier()` function. The range values are taken from `mc_data_sensors`. Those are manually defined ranges based on logger/sensor technical limits and biologically meaningful values.

Value

data.frame with columns:

- sensor_name - name of sensor (e.g., TMS_T1, TMS_moist, HOBO_T) see mc_data_sensors
- min_value - minimal value
- max_value - maximal value
- positive_jump - Maximal difference between two consecutive values. Next value is higher than previous. (Positive number)
- negative_jump - Maximal difference between two consecutive values. Next value is lower than previous. (Positive number)

Examples

```
mc_info_range(mc_data_example_raw)
```

`mc_info_states` *Get states (tags) info table*

Description

This function return data.frame with information about sensor states (tags) see myClim-package

Usage

```
mc_info_states(data)
```

Arguments

`data`	myClim object see myClim-package

Details

This function is useful not only for inspecting actual states (tags) but also as a template for manually manipulating states (tags) in a table editor such as Excel. The output of `mc_info_states()` can be saved as a table, adjusted outside R (adding/removing/modifying rows), and then read back into R to be used as input for mc_states_insert or mc_states_update.

Value

data.frame with columns:

- locality_id - when provided by user then locality ID, when not provided identical with serial number
- logger_index - index of logger in myClim object at the locality
- logger_type - type of logger
- sensor_name - sensor name either original (e.g., TMS_T1, T_C), or calculated/renamed (e.g., "TMS_T1_max", "my_sensor01")
- tag - category of state (e.g., "error", "source", "quality")
- start - start datetime
- end - end datetime
- value - value of tag (e.g., "out of soil", "c:/users/John/tmsData/data_911235678.csv")

Examples

```
mc_info_states(mc_data_example_raw)
```

---

mc_join *Joining time-series from repeated downloads*

Description

The function is designed to merge time-series data obtained through repeated downloads in the same location. Within a specific locality, the function performs the merging based on logger type, physical element, and sensor height

Usage

```
mc_join(data, comp_sensors = NULL)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package
`comp_sensors`	senors for compare and select source logger; If NULL then first is used. (default NULL)

Details

Joining is restricted to the myClim Raw-format (refer to myClim-package). Loggers need to be organized within localities. The simplest method is to use mc_read_data, providing both `files_table` and `localities_table`. When using mc_read_files without metadata, a bit more coding is needed. In this case, you can create multiple myClim objects and specify correct locality names afterwards, then merge these objects using mc_prep_merge, which groups loggers based on identical locality names.

The joining function operates seamlessly without user intervention in two scenarios:

1. when the start of a newer time series aligns with the end of an older one, and
2. when the two time-series share identical values during the overlap.

However, if values differ during the overlap, the user is prompted to interactively choose which time-series to retain and which to discard. myClim provides information about differing time-series in the console, including locality ID, problematic interval (start-end), older logger ID and its time series start-end, and newer logger ID and its time series start-end. Additionally, an interactive graphical window (plotly) displays conflicting time series, allowing the user to zoom in and explore values. In case of multiple conflicts, myClim sequentially asks the user for decisions.

Users have seven options for handling overlap conflicts, six of which are pre-defined. The seventh option allows the user to specify the exact time to trim the older time-series and use the newer one. The options include:

- 1: using the older logger (to resolve this conflict),
- 2: using the newer logger (to resolve this conflict),
- 3: skip this join (same type loggers in locality aren't joined),
- 4: always using the older logger (to resolve this and all other conflicts),
- 5: always using the newer logger (to resolve this and all other conflicts)
- 6: exit joining process.

Users must press the number key, hit Return/Enter, or write in console the exact date in the format `YYYY-MM-DD hh:mm` to trim the older series and continue with the newer series.

Loggers with multiple sensors are joined based on one or more selected sensors (see parameter comp_sensors). The name of the resulting joined sensor is taken from the logger with the oldest data. If serial_number is not equal during logger joining, the resulting serial_number is NA. Clean info is changed to NA except for the step. When joining a non-calibrated sensor with a calibrated one, the calibration information must be empty in the non-calibrated sensor.

For example of joining see myClim vignette.

WARNING

`mc_join` expects a maximum of one logger of a certain type and height measuring certain elements in one locality. In other words, if you use multiple logger of identical type at identical heights in one locality, you cannot use `mc_join` directly; you have to split your locality into sub-localities.

Value

myClim object with joined loggers.

`mc_load` *Load myClim object*

Description

This function loads the myClim .rds data object saved with mc_save. The `mc_save` and `mc_load` functions secure that the myClim object is correctly loaded across myClim versions.

Usage

```
mc_load(file)
```

Arguments

`file`	path to input .rds file

Value

loaded myClim object

Examples

```
tmp_dir <- tempdir()
tmp_file <- tempfile(tmpdir = tmp_dir)
mc_save(mc_data_example_agg, tmp_file)
data <- mc_load(tmp_file)
file.remove(tmp_file)
```

---

mc_LocalityMetadata-class

Class for locality metadata

Description

Class for locality metadata

Details

When reading without metadata, then locality is named after file where the data come from, or after the sensor id where the data come form.

Slots

`locality_id` name of locality

`elevation` of locality

`lat_wgs84` latitude of locality in WGS-84

`lon_wgs84` longitude of locality in WGS-84

`tz_offset` offset from UTC in minutes

`tz_type` type of time zone

`user_data` list for user data

See Also

myClim-package, mc_LoggerMetadata, mc_SensorMetadata `mc_LoggerCleanInfo-class`

Class for logger clean info

Description

Class for logger clean info

Slots

- `step` Time step of microclimatic data series in seconds
- `count_duplicities` count of duplicated records - values with same date
- `count_missing` count of missing records; Period between the records should be the same length. If not, than missing.
- `count_disordered` count of records incorrectly ordered in time. In table, newer record is followed by the older.
- `rounded` T/F indication whether myClim automatically rounded time series to the closes half (HH:00, HH:30) e.g. 13:07 -> 13:00

`mc_LoggerMetadata-class`

Class for logger metadata

Description

Class for logger metadata

Slots

- `type` of logger (TMS, Thermo, Dendro, HOBO)
- `serial_number` serial number of the logger
- `step` time step of microclimatic time-seris in seconds. When provided by user, is used in `mc_prep_clean()` function instead of automatic step detection

mc_MainMetadata-class *Class for myClim object metadata*

Description

Class for myClim object metadata

Slots

`version` the version of the myClim package in which the object was created

`format_type` type of format (Raw-format, Agg-format)

See Also

myClim-package

mc_MainMetadataAgg-class *Class for myClim object metadata in Agg-format*

Description

Class for myClim object metadata in Agg-format

Slots

`version` the version of the myClim package in which the object was created

`format_type` type of format (Raw-format, Agg-format)

`step` time step of data in seconds

`period` value from `mc_agg()` (e.g. month, day, all...)

`intervals_start` start datetime of data intervals for spacial periods all and custom (see `mc_agg()`)

`intervals_end` end datetime of data intervals for spacial periods all and custom (see `mc_agg()`)

See Also

mc_MainMetadata myClim-package

`mc_Physical-class` *Class for physical*

Description

Class defining the element of the records (temperature, volumetric water content, height...)

Details

See e.g. definition of temperature. Similarly as the definition of new loggers, new physicals can be added like modules.

```
Slot "name": "T_C"
Slot "description": "Temperature °C"
Slot "units": "°C"
Slot "viridis_color_map": "C"
Slot "scale_coeff": 0.03333333
```

Slots

`name` of physical

`description` character info

`units` measurument (°C, %, m3/m3, raw, mm, ...)

`viridis_color_map` viridis color map option

`scale_coeff` coefficient for plot; value * scale_coef is in range 0-1

See Also

mc_data_physical

`mc_plot_image` *Plot data - image*

Description

Function plots single sensor form myClim data into PNG file with image() R base function. This was designed for fast, and easy data visualization especially focusing on missing values visualization and general data picture.

Usage

```
mc_plot_image(
  data,
  filename,
  title = "",
  localities = NULL,
  sensors = NULL,
  height = 1900,
  left_margin = 12,
  use_utc = TRUE
)
```

Arguments

`data`	myClim object see myClim-package
`filename`	output file name (file path)
`title`	of plot; default is empty
`localities`	names of localities; if NULL then all (default NULL)
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`height`	of image; default = 1900
`left_margin`	width of space for sensor_labels; default = 12
`use_utc`	if FALSE, then the time shift from `tz_offset` metadata is used to correct (shift) the output time-series (default TRUE) In the Agg-format myClim object `use_utc = FALSE` is allowed only for steps shorter than one day. In myClim the day nd longer time steps are defined by the midnight, but this represent whole day, week, month, year... shifting daily, weekly, monthly... data (shift midnight) does not make sense in our opinion. But when user need more flexibility, then myClim Raw-format can be used, In Raw-format `use_utc` is not limited, user can shift an data without the restrictions. See myClim-package

Details

Be careful with bigger data. Can take some time.

Value

PNG file created as specified in output file name

Examples

```
tmp_dir <- tempdir()
tmp_file <- tempfile(tmpdir = tmp_dir)
mc_plot_image(mc_data_example_clean, tmp_file, "T1 sensor", sensors="TMS_T1")
file.remove(tmp_file)
```

mc_plot_line *Plot data - ggplot2 geom_line*

Description

Function plots data with ggplot2 geom_line. Plot is returned as ggplot faced grid and is optimized for saving as facet, paginated PDF file.

Usage

```
mc_plot_line(
  data,
  filename = NULL,
  sensors = NULL,
  scale_coeff = NULL,
  png_width = 1900,
  png_height = 1900,
  start_crop = NULL,
  end_crop = NULL,
  use_utc = TRUE,
  localities = NULL,
  facet = "locality",
  color_by_logger = FALSE
)
```

Arguments

`data`	myClim object see myClim-package
`filename`	output file name/path with the extension - supported formats are .pdf and .png (default NULL) If NULL then the plot is displayed and can be returned into r environment but is not saved to file.
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`scale_coeff`	scale coefficient for secondary axis (default NULL)
`png_width`	width for png output (default 1900)
`png_height`	height for png output (default 1900)
`start_crop`	POSIXct datetime in UTC for crop data (default NULL)
`end_crop`	POSIXct datetime in UTC for crop data (default NULL)
`use_utc`	if FALSE, then the time shift from `tz_offset` metadata is used to correct (shift) the output time-series (default TRUE) In the Agg-format myClim object `use_utc = FALSE` is allowed only for steps shorter than one day. In myClim the day nd longer time steps are defined by the midnight, but this represent whole day, week, month, year... shifting daily, weekly, monthly... data (shift midnight) does not make sense in our opinion. But

when user need more flexibility, then myClim Raw-format can be used, In Raw-format `use_utc` is not limited, user can shift an data without the restrictions. See myClim-package

`localities` names of localities; if NULL then all (default NULL)

`facet` possible values (`NULL`, `"locality"`, `"physical"`)

- `facet = "locality"` each locality is plotted (default) in separate plot in R and separate row in PDF if filename.pdf is provided.
- `facet = "physical"` sensors with identical physical (see mc_data_physical) are grouped together across localities.
- `facet = NULL`, all localities and sensors (max 2 physicals, see details) are plotted in single plot

`color_by_logger`
If TRUE, the color is assigned by logger to differentiate individual loggers (random colors) if false, the color is assigned by physical. (default FALSE)

Details

Saving as the PDF file is recommended, because the plot is optimized to be paginate PDF (facet line plot is distributed to pages), each locality can be represented by separate plot (`facet = "locality"`) default, which is especially useful for bigger data. When `facet = NULL` then single plot is returned showing all localities together. When `facet = physical` sensors with identical physical units are grouped together across localities. Maximal number of physical units (elements) of sensors to be plotted in one plot is two. First element is related to primary and second to secondary y axis. In case, there are multiple sensors with identical physical on one locality, they are plotted together for `facet = "locality"` e.g., when you have TMS_T1, TMS_T2, TMS_T3, Thermo_T, and VWC you get plot with 5 lines of different colors and two y axes. Secondary y axes are scaled with calculation `values * scale_coeff`. If scaling coefficient is NULL than function try to detects scale coefficient from physical unit of sensors see mc_Physical. Scaling is useful when plotting together e.g. temperature and moisture. For native myClim loggers (TOMST, HOBO U-23) scaling coefficients are pre-defined. For other cases when plotting two physicals together, it is better to set scaling coefficients by hand.

Value

ggplot2 object

Examples

```
tms.plot <- mc_filter(mc_data_example_agg, localities = "A6W79")
p <- mc_plot_line(tms.plot,sensors = c("TMS_T3","TMS_T1","TMS_moist"))
p <- p+ggplot2::scale_x_datetime(date_breaks = "1 week", date_labels = "%W")
p <- p+ggplot2::xlab("week")
p <- p+ggplot2::scale_color_manual(values=c("hotpink","pink", "darkblue"),name=NULL)
```

mc_plot_loggers *Plot data from loggers*

Description

Function save separate files (*.png) per the loggers to the directory. Only Raw-format supported, Agg-format not supported. For Agg-format use `mc_plot_line()`. Function was primary designed for Tomst TMS loggers for fast, and easy data visualization.

Usage

```
mc_plot_loggers(
  data,
  directory,
  localities = NULL,
  sensors = NULL,
  crop = c(NA, NA)
)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package
`directory`	path to output directory
`localities`	names of localities; if NULL then all (default NULL)
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`crop`	datetime range for plot, not cropping if NA (default c(NA, NA))

Value

PNG files created in the output directory

Examples

```
tmp_dir <- file.path(tempdir(), "plot")
mc_plot_loggers(mc_data_example_clean, tmp_dir)
unlink(tmp_dir, recursive=TRUE)
```

`mc_plot_raster` *Plot data - ggplot2 geom_raster*

Description

Function plots data with ggplot2 geom_raster. Plot is returned as ggplot faced raster and is primary designed to be saved as .pdf file (recommended) or .png file. Plotting into R environment without saving any file is also possible. See details.

Usage

```
mc_plot_raster(
  data,
  filename = NULL,
  sensors = NULL,
  by_hour = TRUE,
  png_width = 1900,
  png_height = 1900,
  viridis_color_map = NULL,
  start_crop = NULL,
  end_crop = NULL,
  use_utc = TRUE
)
```

Arguments

`data` myClim object see myClim-package

`filename` output with the extension - supported formats are .pdf and .png (default NULL) If NULL then the plot is shown/returned into R environment as ggplot object, but not saved to file.

`sensors` names of sensor; should have same physical unit see `names(mc_data_sensors)`

`by_hour` if TRUE, then y axis is plotted as an hour, else original time step (default TRUE)

`png_width` width for png output (default 1900)

`png_height` height for png output (default 1900)

`viridis_color_map`
viridis color map option; if NULL, then used value from mc_data_physical

- "A" - magma
- "B" - inferno
- "C" - plasma
- "D" - viridis
- "E" - cividis
- "F" - rocket
- "G" - mako
- "H" - turbo

start_crop — POSIXct datetime in UTC for crop data (default NULL)

end_crop — POSIXct datetime in UTC for crop data (default NULL)

use_utc — if FALSE, then the time shift from `tz_offset` metadata is used to correct (shift) the output time-series (default TRUE)

In the Agg-format myClim object `use_utc = FALSE` is allowed only for steps shorter than one day. In myClim the day nd longer time steps are defined by the midnight, but this represent whole day, week, month, year... shifting daily, weekly, monthly... data (shift midnight) does not make sense in our opinion. But when user need more flexibility, then myClim Raw-format can be used, In Raw-format `use_utc` is not limited, user can shift an data without the restrictions. See myClim-package

Details

Saving as the .pdf file is recommended, because the plot is optimized to be paginate PDF (facet raster plot is distributed to pages), which is especially useful for bigger data. In case of plotting multiple sensors to PDF, the facet grids are grouped by sensor. I.e., all localities of sensor_1 followed by all localities of sensor_2 etc. When plotting only few localities, but multiple sensors, each sensor has own page. I.e., when plotting data from one locality, and 3 sensors resulting PDF has 3 pages. In case of plotting PNG, sensors are plotted in separated images (PNG files) by physical. I.e, when plotting 3 sensors in PNG it will save 3 PNG files named after sensors. Be careful with bigger data in PNG. Play with `png_height` and `png_width`. When too small height/width, image does not fit and is plotted incorrectly. Plotting into R environment instead of saving PDF or PNG is possible, but is recommended only for low number of localities (e.g. up to 10), because high number of localities plotted in R environment results in very small picture which is hard/impossible to read.

Value

list of ggplot2 objects

Examples

```
tmp_dir <- tempdir()
tmp_file <- tempfile(tmpdir = tmp_dir, fileext=".pdf")
mc_plot_raster(mc_data_example_agg, filename=tmp_file, sensors=c("TMS_T3","TM_T"))
file.remove(tmp_file)
```

---

mc_prep_calib *Sensors calibration*

Description

This function calibrate values of sensor (microclimatic records) using the myClim object `sensor$calibration` parameters provided by `mc_prep_calib_load()`. Microclimatic records are changed and myClim object parameter `sensor$metadata@calibrated` is set to TRUE. It isn't allowed to calibrate sensor multiple times.

Usage

```
mc_prep_calib(data, localities = NULL, sensors = NULL)
```

Arguments

`data`	myClim object in Raw-format or Agg-format having calibration data in metadata slot `sensor$calibration`
`localities`	vector of locality_ids where to perform calibration, if NULL, then calibrate sensors on all localities (default NULL)
`sensors`	vector of sensor names where to perform calibration see `names(mc_data_sensors)`; if NULL, then calibrate all sensors having calibration parameters loaded (default NULL)

Details

This function performs calibration itself. It uses the calibration values (cor_factor, cor_slope) stored in myClim object sensor metadata sensor calibration loaded with `mc_prep_calib_load()`. As it is possible to have multiple calibration values for one sensor in time (re-calibration after some time) different calibration values can be applied based on the calibration time. Older microclimatic records then first calibration `datetime` available are calibrated anyway (in case sensor was calibrated ex-post) with the first calibration parameters available.

This function is not designed for moisture_raw calibration (conversion to volumetric water content) for this use `mc_calc_vwc()`

Only sensors with real value type can be calibrated. see `mc_data_sensors()`

Value

same myClim object as input but with calibrated sensor values.

`mc_prep_calib_load` *Load sensor calibration parameters to correct microclimatic records*

Description

This function loads calibration parameters from data.frame *logger_calib_table* and stores them in the myClim object metadata. This function does not calibrate data. For calibration itself run `mc_prep_calib()`

Usage

```
mc_prep_calib_load(data, calib_table)
```

Arguments

`data`	myClim object in Raw-format. see myClim-package
`calib_table`	data.frame with columns (serial_number, sensor_id, datetime, slope, intercept)

Details

This function allows user to provide correction coefficients `cor_factor` and `cor_slope` for linear sensor calibration. Calibrated data have by default the form of linear function terms:
`calibrated value = original value * (cor_slope + 1) + cor_factor`

In case of one-point calibration, `cor_factor` can be estimated as: `cor_factor = reference value - sensor value` and `cor_slope` should be set to 0. This function loads sensor-specific calibration coefficients from *calib_table* and stores them into myClim Raw-format object metadata. The *calib_table* is data.frame with 5 columns:

- serial_number = serial number of the logger
- sensor_id = name of sensor, e.g. "TMS_T1"
- datetime = the date of the calibration in POSIXct type
- cor_factor = the correction factor
- cor_slope = the slope of calibration curve (in case of one-point calibration, use cor_slope = 0)

It is not possible to change calibration parameters for already calibrated sensor. This prevents repeated calibrations. Once `mc_prep_calib()` is called then it is not allowed to provide new calibration data, neither run calibration again.

Value

myClim object with loaded calibration information in metadata. Microclimatic records are not calibrated, only ready for calibration. To calibrate records run `mc_prep_calib()`

`mc_prep_clean` *Cleaning datetime series*

Description

By default, `mc_prep_clean` runs automatically when `mc_read_files()` or `mc_read_data()` are called. `mc_prep_clean` checks the time-series in the myClim object in Raw-format for missing, duplicated, and disordered records. The function can either directly regularize microclimatic time-series to a constant time-step, remove duplicated records, and fill missing values with NA (`resolve_conflicts=TRUE`); or it can insert new states (tags) see mc_states_insert to highlight records with conflicts i.e. duplicated datetime but different measurement values (`resolve_conflicts=FALSE`) but not perform the cleaning itself. When there were no conflicts, cleaning is performed in both cases (`resolve_conflicts=TRUE or FALSE`) See details.

Usage

```
mc_prep_clean(data, silent = FALSE, resolve_conflicts = TRUE)
```

Arguments

data — myClim object in Raw-format. see myClim-package

silent — if true, then cleaning log table and progress bar is not printed in console (default FALSE), see `mc_info_clean()`

resolve_conflicts — by default the object is automatically cleaned and conflict measurements with closest original datetime to rounded datetime are selected, see details. (default TRUE) If FALSE and conflict records exist the function returns the original, uncleaned object with tags (states) "conflict" highlighting records with duplicated datetime but different measurement values.When conflict records does not exist, object is cleaned in both TRUE and FALSE cases.

Details

The function `mc_prep_clean` can be used in two different ways depending on the parameter `resolve_conflicts`. When `resolve_conflicts=TRUE`, the function performs automatic cleaning and returns a cleaned myClim object. When `resolve_conflicts=FALSE`, and myClim object contains conflicts, the function returns the original, uncleaned object with tags (states) see mc_states_insert highlighting records with duplicated datetime but different measurement values. When there were no conflicts, cleaning is performed in both cases (`resolve_conflicts=TRUE OR FALSE`)

Processing the data with `mc_prep_clean` and resolving the conflicts is a mandatory step required for further data handling in the `myClim` library.

This function guarantee that all time series are in chronological order, have regular time-step and no duplicated records. Function `mc_prep_clean` use either time-step provided by user during data import with `mc_read` (used time-step is permanently stored in logger metadata mc_LoggerMetadata; or if time-step is not provided by the user (NA),than myClim automatically detects the time-step from input time series based on the last 100 records. In case of irregular time series, function returns warning and skip the series.

In case the time-step is regular, but is not nicely rounded, function rounds the time series to the closest nice time and shifts original data. E.g., original records in 10 min regular step c(11:58, 12:08, 12:18, 12:28) are shifted to newly generated nice sequence c(12:00, 12:10, 12:20, 12:30). Note that microclimatic records are not modified but only shifted. Maximum allowed shift of time series is 30 minutes. For example, when the time-step is 2h (e.g. 13:33, 15:33, 17:33), the measurement times are shifted to (13:30, 15:30, 17:30). When you have 2h time step and wish to go to the whole hour (13:33 -> 14:00, 15:33 -> 16:00) the only way is aggregation - use `mc_agg(period="2 hours")` command after data cleaning.

In cases when the user provides a time-step during data import in `mc_read` functions instead of relying on automatic step detection, and the provided step does not correspond with the actual records (i.e., the logger records data every 900 seconds but the user provides a step of 3600 seconds), the myClim rounding routine consolidates multiple records into an identical datetime. The resulting value corresponds to the one closest to the provided step (i.e., in an original series like ...9:50, 10:05, 10:20, 10:35, 10:50, 11:05..., the new record would be 10:00, and the value will be taken from the original record at 10:05). This process generates numerous warnings in `resolve_conflicts=TRUE` and a multitude of tags in `resolve_conflicts=FALSE`.

Value

- cleaned myClim object in Raw-format (default) `resolve_conflicts=TRUE` or `resolve_conflicts=FALSE` but no conflicts exist
- cleaning log is by default printed in console, but can be called also later by `mc_info_clean()`
- non cleaned myClim object in Raw-format with "conflict" tags `resolve_conflicts=FALSE` and conflicts exist

Examples

```
cleaned_data <- mc_prep_clean(mc_data_example_raw)
```

`mc_prep_crop` *Crop datetime*

Description

This function crop data by datetime

Usage

```
mc_prep_crop(
  data,
  start = NULL,
  end = NULL,
  localities = NULL,
  end_included = TRUE
)
```

Arguments

`data`	myClim object see myClim-package
`start`	optional; POSIXct datetime **in UTC**; single value or vector; start datetime is included (default NULL)
`end`	optional, POSIXct datetime **in UTC**; single value or vector (default NULL)
`localities`	vector of locality_ids to be cropped; if NULL then all localities are cropped (default NULL)
`end_included`	if TRUE then end datetime is included (default TRUE), see details

Details

Function is able to crop data from `start` to end but works also with `start` only and end only. When only `start` is provided, then function crops only the beginning of the tim-series and vice versa with end.

If `start` or `end` is a single POSIXct value, it is used for all or selected localities (regular crop). However, if `start` and `end` are vectors of POSIXct values with the same length as the localities vector, each locality is cropped by its own time window (irregular crop).

The end_included parameter is used for selecting, whether to return data which contains end time or not. For example when cropping the data to rounded days, typically users use midnight. 2023-06-15 00:00:00 UTC. But midnight is the last date of ending day and the same time first date of the next day. Thus, there will be the last day with single record. This can be confusing in aggregation (e.g. daily mean of single record per day, typically NA) so sometimes it is better to exclude end and crop on 2023-06-14 23:45:00 UTC (15 minutes records).

Value

cropped data in the same myClim format as input.

Examples

```
cropped_data <- mc_prep_crop(mc_data_example_clean, end=as.POSIXct("2020-02-01", tz="UTC"))
```

`mc_prep_fillNA` *Fill NA*

Description

This function approximate NA (missing) values. It was designed to fill only small gaps in microclimatic time-series therefore, the default maximum length of the gap is 5 missing records and longer gaps are not filled Only linear method is implemented from zoo::na.approx function.

Usage

```
mc_prep_fillNA(
  data,
  localities = NULL,
  sensors = NULL,
  maxgap = 5,
  method = "linear"
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`localities`	names of localities; if NULL then all (default NULL)
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`maxgap`	maximum number of consecutively NA values to fill (default 5)
`method`	used for approximation. It is implemented now only "linear". (default "linear")

Value

myClim object with filled NA values

mc_prep_merge *Merge myClim objects*

Description

This function is designed to merge more existing myClim objects into one.

Usage

```
mc_prep_merge(data_items)
```

Arguments

`data_items`	list of myClim objects see myClim-package; Format (Raw/Agg) of merged objects must be same.

Details

This function works only when the input myClim objects have the same format (Raw-format, Agg-format) It is not possible to merge Raw wit Agg format. Identical time-step is required for Agg-format data.

When the merged myClim objects in Raw-format contains locality with same names (locality_id), than list of loggers are merged on the locality. Sensors with the same name does not matter here. Loggers with the same name within the locality are allowed in the Raw-format.

When the merged myClim objects in Agg-format contains locality with same names (locality_id). than the sensors are merged on the locality. Sensors with same names are renamed.

Value

merged myClim object in the same format as input objects

Examples

```
merged_data <- mc_prep_merge(list(mc_data_example_raw, mc_data_example_raw))
```

mc_prep_meta_locality *Set metadata of localities*

Description

This function allows you to add or modify locality metadata including locality names. See mc_LocalityMetadata. You can import metadata from named list or from data frame. See details.

Usage

```
mc_prep_meta_locality(data, values, param_name = NULL)
```

Arguments

`data`	myClim object see myClim-package
`values`	for localities can be named list or table • named list: `metadata <- list(locality_id=value)`; `param_name` must be set • table with column `locality_id` and another columns named by metadata parameter name; to rename locality use `new_locality_id`. Parameter `param_name` must be NULL.
`param_name`	name of locality metadata parameter; Default names are `locality_id`, `elevation`, `lat_wgs84`, `lon_wgs84`, `tz_offset`. Another names are inserted to `user_data` list. see mc_LocalityMetadata

Details

Locality metadata is critical e.g. for correctly handling time zones. By providing geographic coordinates in locality metadata, the user can later harmonize all data to the local solar time (midday) #' with `mc_prep_solar_tz()` or calculate temporal offset to the UTC base on local time-zone. Alternatively, the user can directly provide the offset (in minutes) for individual localities. This can be useful especially for heterogeneous data sets containing various localities with loggers recording in local time. By providing temporal offset for #' each locality separately, you can unify the whole dataset to UTC. Note that when tz_offset is set manually, than tz_type is set to `user defined`.

For minor metadata modification it is practical to use named list in combination with `param_name` specification. E.g. when you wish to modify only time zone offset, then set `param_name="tz_offset"` and provide named list with locality name and offset value `list(A1E05=60)`. Similarly, you can modify other metadata slots mc_LocalityMetadata.

For batch or generally more complex metadata modification you can provide data.frame with columns specifying `locality_id` and one of `new_locality_id, elevation, lat_wgs84, lon_wgs84, tz_offset`. Provide locality_id (name) and the value in column of metadata you wish to update. In case of using data.frame use `param_name = NULL`

Value

myClim object in the same format as input, with updated metadata

Examples

```
data <- mc_prep_meta_locality(mc_data_example_raw, list(A1E05=60), param_name="tz_offset")
```

`mc_prep_meta_sensor` *Set metadata of sensors*

Description

This function allows you to modify sensor metadata including sensor name. See mc_SensorMetadata

Usage

```
mc_prep_meta_sensor(
  data,
  values,
  param_name,
  localities = NULL,
  logger_types = NULL
)
```

Arguments

data	myClim object see myClim-package
values	named list with metadata values; names of items are sensor_names e.g. for changing sensor height use `list(TMS_T1="soil 8 cm")`
param_name	name of the sensor metadata parameter you want to change; You can change `name` and `height` of sensor.
localities	optional filter; vector of `locality_id` where to change sensor metadata; if NULL than all localities (default NULL)
logger_types	optional filter; vector of `logger_type` where to change metadata; if NULL than all logger types (default NULL); `logger_type`is useful only for Raw-format of myClim having the level of logger see myClim-package

Value

myClim object in the same format as input, with updated sensor metadata

Examples

```
data <- mc_prep_meta_sensor(mc_data_example_raw, list(TMS_T1="my_TMS_T1"), param_name="name")
```

---

mc_prep_solar_tz *Set solar time offset against UTC time*

Description

This function calculates the temporal offset between local solar time and UTC time zone. Calculation is based on geographic coordinates of each locality. Therefore, the function does not work when longitude coordinate is not provided.

Usage

```
mc_prep_solar_tz(data)
```

Arguments

data	myClim object see myClim-package

Details

myClim assumes that the data are in UTC. To calculate temporal offset based on local solar time, this function requires geographic coordinates (at least longitude) to be provided in locality metadata slot `lon_wgs84` (in decimal degrees). Geographic coordinates for each locality can be provided already during data reading, see `mc_read_data()`, or added later with `mc_prep_meta_locality()` function.

TZ offset (in minutes) is calculated as `longitude / 180 * 12 * 60`.

Value

myClim object in the same format as input, with `tz_offset` filled in locality metadata

Examples

```
data_solar <- mc_prep_solar_tz(mc_data_example_clean)
```

`mc_prep_TMSoffsoil` *Detection of out-of-soil measurements from TMS logger*

Description

This function creates new virtual sensor labelling anomalies in TMS logger caused by displacement out of from soil.

Usage

```
mc_prep_TMSoffsoil(
  data,
  localities = NULL,
  soil_sensor = mc_const_SENSOR_TMS_T1,
  air_sensor = mc_const_SENSOR_TMS_T2,
  moist_sensor = mc_const_SENSOR_TMS_moist,
  output_sensor = "off_soil",
  smooth = FALSE,
  smooth_window = 10,
  smooth_threshold = 0.5,
  sd_threshold = 0.76085,
  minmoist_threshold = 721.5
)
```

Arguments

`data`	cleaned myClim object see myClim-package
`localities`	names of localities; if NULL then all (default NULL)
`soil_sensor`	character, soil temperature sensor (default `mc_const_SENSOR_TMS_T1`)
`air_sensor`	character, air temperature sensor (default `mc_const_SENSOR_TMS_T2`)

`moist_sensor` character, soil moisture sensor (default `mc_const_SENSOR_TMS_moist`)

`output_sensor` character, name of virtual sensor to store ouptup values (default "off_soil")

`smooth` logical, smooth out isolated faulty/correct records using floating window (default FALSE)

`smooth_window` integer, smooth floating window width (in days) (default 10)

`smooth_threshold` numeric, floating window threshold for detection of faulty records. (default 0.5)

`sd_threshold` numeric, threshold value for the criteria on the ratio of standard deviation of the soil sensor to the above-ground sensor temperatures (default 0.76085)

`minmoist_threshold` numeric, threshold value for criteria on the minimum soil moisture (default 721.5)

Details

TMS loggers, when correctly installed in the soil, exhibit certain temperature and soil moisture signal characteristics. Temperature varies the most at the soil interface, and temperature fluctuations in the soil are minimized. The moisture signal from a sensor that has lost direct contact with the soil is reduced. The following criteria are used for detecting faulty measurements: the ratio of the standard deviations of the soil sensor to the above-ground sensor within 24h moving window is greater than the defined threshold (default 0.76085), and simultaneously, the soil moisture minimum within 24h mowing window is less than 721.5. Optionally, the prediction results can be smoothed using a floating window to average-out unlikely short periods detected by the algorithm. Selection and parametrization of criteria was done using a recursive partitioning (rpart::rpart) on the training set of 7.8M readings in 154 TMS timeseries from different environmental settings (temperate forests, tropical rainforest, cold desert, alpine and subnival zone, and invalid measurements from loggers stored in the office or displaced from the soil). Sensitivity of the method (true positive rate) on was 95.1% and specificity (true negative rate) was 99.4% using function default parameters. Smoothing with 10 day floating window increased sensitivity to 96.8% while retaining specifity at the same level of 99.4%. Decreasing 'smooth_threshold' below 0.5 will extend periods flagged as faulty measurement.

Value

numeric vector (0 = correct measurement, 1 = faulty measurement) stored as virtual sensor in myClim object

Examples

```
data <- mc_read_files(system.file("extdata", "data_93142760_201904.csv", package = "myClim"),
                      "TOMST")
data <- mc_prep_TMSoffsoil(data)
mc_plot_line(data, sensors = c("off_soil","TMS_T1", "TMS_T2","TMS_T3"))
```

`mc_read_data` *Reading files with locality metadata*

Description

This function has two tables as the parameters.

(i) `files_table` is required parameter, it ust contain *paths* pointing to raw csv logger files, specification of *data format* (logger type) and *locality name*.

(ii) `localities_table` is optional, containing *locality id* and metadata e.g. longitude, latitude, elevation...

Usage

```
mc_read_data(
  files_table,
  localities_table = NULL,
  clean = TRUE,
  silent = FALSE,
  user_data_formats = NULL
)
```

Arguments

`files_table` path to csv file or data.frame object see example with 3 required columns and few optional:

required columns:

- path - path to files
- locality_id - unique locality id
- data_format see mc_data_formats, `names(mc_data_formats)`

optional columns:

- serial_number - logger serial number. If is NA, than myClim tries to detect serial number from file name (for TOMST) or header (for HOBO)
- logger_type - type of logger. This defines individual sensors attributes (measurement heights and physical units) of the logger. Important when combining the data from multiple loggers on the locality. If not provided, myClim tries to detect loger_type from the source data file structure (applicable for HOBO, Dendro, Thermo and TMS), but automatic detection of TMS_L45 is not possible. Pre-defined logger types are: ("Dendro", "HOBO", "Thermo", "TMS", "TMS_L45") Default heights of sensor based on logger types are defined in table mc_data_heights
- date_format A character vector specifying the custom date format(s) for the `lubridate::parse_date_time()` function (e.g., "%d.%m.%Y %H:%M:%S"). Multiple formats can be defined either in in CSV or in R data.frame using `@` character as separator (e.g., "%d.%m.%Y %H:%M:%S@%Y.%m.%d %H:%M:%S"). The first matching format will be selected for parsing, multiple formats are applicable to single file.

- tz_offset - If source datetimes aren't in UTC, then is possible define offset from UTC in minutes. Value in this column have the highest priority. If NA then auto detection of timezone in files. If timezone can't be detected, then UTC is supposed. Timezone offset in HOBO format can be defined in header. In this case function try detect offset automatically. Ignored for TOMST TMS data format (they are always in UTC)
- step - Time step of microclimatic time-series in seconds. When provided, then used in mc_prep_clean instead of automatic step detection. See details.

`localities_table`
: path to csv file ("c:/user/localities.table.csv") or R data.frame see example. Localities table is optional (default NULL). The `locality_id` is the only required column. Other columns are optional. Column names corresponding with the myclim pre-defined locality metadata (elevation, lon_wgs84, lat_wgs84, tz_offset) are associted withthose pre-defined metadata slots, other columns are written into `metadata@user_data` myClim-package.

  required columns:

  - locality_id - unique locality id

  optional columns:

  - elevation - elevation (in m)
  - lon_wgs84 - longitude (in decimal degrees)
  - lat_wgs84 - latitude (in decimal degrees)
  - tz_offset - locality time zone offset from UTC, applicable for converting time-series from UTC to local time.
  - ... - any other columns are imported to `metadata@user_data`

`clean`
: if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)

`silent`
: if TRUE, then any information is not printed in console (default FALSE)

`user_data_formats`
: custom data formats; use in case you have your own logger files not pre-defined in myClim - list(key=mc_DataFormat) mc_DataFormat (default NULL)

  If custom data format is defined the key can be used in data_format parameter in `mc_read_files()` and `mc_read_data()`. Custom data format must be defined first, and then an be used for reading.

Details

The input tables could be R data.frames or csv files. When loading `files_table` and `localities_table` from external CSV they must have header, column separator must be comma ",". If you only need to place loggers to correct localities, `files_table` is enough. If you wish to provide localities additional metadata, you need also `localities_table`

By default, data are cleaned with the function mc_prep_clean see function description. mc_prep_clean detects gaps in time-series data, duplicated records, or records in the wrong order. Importantly, mc_prep_clean also applies a **step parameter** if provided. The step parameter can be used either instead of automatic step detection which can sometime failed, or to prune microclimatic data. For example, if you have a 15-minute time series but you wish to keep only one record per hour (without aggregating), you can use step parameter. However, if a step is provided and `clean = FALSE`, then the step is only stored in the metadata of myClim, and the time-series data is not cleaned, and the step is not applied.

Value

myClim object in Raw-format see myClim-package

See Also

mc_DataFormat

Examples

```
files_csv <- system.file("extdata", "files_table.csv", package = "myClim")
localities_csv <- system.file("extdata", "localities_table.csv", package = "myClim")
tomst_data <- mc_read_data(files_csv, localities_csv)
```

---

mc_read_files *Reading files or directories*

Description

This function read one or more CSV/TXT files or directories of identical, pre-defined logger type (format) see mc_DataFormat and mc_data_formats. This function does not support loading locality or sensor metadata while reading. Metadata can be loaded through `mc_read_data()` or can be provided later with function `mc_prep_meta_locality()`

Usage

```
mc_read_files(
  paths,
  dataformat_name,
  logger_type = NA_character_,
  recursive = TRUE,
  date_format = NA_character_,
  tz_offset = NA_integer_,
  step = NA_integer_,
  clean = TRUE,
  silent = FALSE,
  user_data_formats = NULL
)
```

Arguments

- `paths` — vector of paths to files or directories
- `dataformat_name` — data format of logger; one of `names(mc_data_formats)`
- `logger_type` — type of logger (default NA), can be one of pre-defined see `mc_read_data()` or any custom string
- `recursive` — recursive search in sub-directories (default TRUE)
- `date_format` — format of date in your hobo files e.g. "%d.%m.%y %H:%M:%S" (default NA). TOMST TMS files used to have stable date format, therefore this parameter may be omitted for TMS files because myClim will try to detect one of formerly stable formats, but nowadays user can adjust any date format also for TMS. For other loggers this parameter is required. You can provide multiple formats to by tried, multiple formats can be combined for reading single file. e.g. c("%d.%m.%Y %H:%M:%S", "%Y.%m.%d %H:%M", "%d.%m.%Y")
- `tz_offset` — timezone offset in minutes; It is required only for non-UTC data (custom settings in HOBO). Not used in TMS (default NA)
- `step` — time step of microclimatic time-series in seconds. When provided, then is used in mc_prep_clean instead of automatic step detection. See details. If not provided (NA), is automatically detected in mc_prep_clean. (default NA)
- `clean` — if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)
- `silent` — if TRUE, then any information is not printed in console (default FALSE)
- `user_data_formats` — custom data formats; use in case you have your own logger files not pre-defined in myClim - list(key=mc_DataFormat) mc_DataFormat (default NULL)

  If custom data format is defined the key can be used in data_format parameter in `mc_read_files()` and `mc_read_data()`. Custom data format must be defined first, and then an be used for reading.

Details

If file is not in expected format, then file is skipped and warning printed in console. CSV/TXT files (loggers raw data) are in resulting myClim object placed to separate localities with empty metadata. Localities are named after serial_number of logger. Pre-defined logger types are ("Dendro","HOBO","Thermo","TMS","TMS_L45")

By default, data are cleaned with the function mc_prep_clean see function description. mc_prep_clean detects gaps in time-series data, duplicated records, or records in the wrong order. Importantly, mc_prep_clean also applies a **step parameter** if provided. The step parameter can be used either instead of automatic step detection which can sometime failed, or to prune microclimatic data. For example, if you have a 15-minute time series but you wish to keep only one record per hour (without aggregating), you can use step parameter. However, if a step is provided and `clean = FALSE`, then the step is only stored in the metadata of myClim, and the time-series data is not cleaned, and the step is not applied.

It is good to specify `date_formatas` this can often be the reason why reading have failed (see warnings after reading).

Value

myClim object in Raw-format see myClim-package

See Also

mc_DataFormat, `mc_prep_clean()`

Examples

```
files <- c(system.file("extdata", "data_91184101_0.csv", package = "myClim"),
           system.file("extdata", "data_94184102_0.csv", package = "myClim"))
tomst_data <- mc_read_files(files, "TOMST",
                            date_format = c("%d.%m.%Y %H:%M:%S",
                                            "%Y.%m.%d %H:%M",
                                            "%d.%m.%Y"))

# user_data_formats
files <- system.file("extdata", "TMS94184102.csv", package = "myClim")
user_data_formats <- list(my_logger=new("mc_DataFormat"))
user_data_formats$my_logger@date_column <- 2
user_data_formats$my_logger@date_format <- "%Y-%m-%d %H:%M:%S"
user_data_formats$my_logger@tz_offset <- 0
user_data_formats$my_logger@columns[[mc_const_SENSOR_T_C]] <- c(3, 4, 5)
user_data_formats$my_logger@columns[[mc_const_SENSOR_real]] <- 6
my_data <- mc_read_files(files, "my_logger", silent=TRUE, user_data_formats=user_data_formats)
```

`mc_read_long` *Reading data from long data.frame*

Description

This is universal function designed to read time series and values from long data.frame to myClim object.

Usage

```
mc_read_long(data_table, sensor_ids = list(), clean = TRUE, silent = FALSE)
```

Arguments

`data_table` long data.frame with Columns:

- locality_id - character; id of locality
- sensor_name - can be any character string, recommended are these: `names(mc_data_sensors)`
- datetime - POSIXct in UTC timezone is required
- value

`sensor_ids`	list with relations between sensor_names and sensor_ids (default list()); sensor_id is key from `names(mc_data_sensors)`. E.g., `sensor_ids <- list(precipitation="real", maxAirT="T_C")` If sensor_name is the same as sensor_id does not have to be provided.
`clean`	if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)
`silent`	if TRUE, then any information is not printed in console (default FALSE)

Details

Similar like mc_read_wide but is capable to read multiple sensors from single table. Useful for data not coming from supported microclimatic loggers. E.g. meteorological station data. By default data are cleaned with function `mc_prep_clean()`.

Value

myClim object in Raw-format

See Also

mc_read_wide

`mc_read_tubedb` *Reading data from TubeDB*

Description

Function is reading data from TubeDB (https://environmentalinformatics-marburg.github.io/tubedb/) into myClim object.

Usage

```
mc_read_tubedb(
  tubedb,
  region = NULL,
  plot = NULL,
  sensor_ids = NULL,
  clean = TRUE,
  silent = FALSE,
  aggregation = "raw",
  quality = "no",
  ...
)
```

Arguments

`tubedb`	object for connection to server see rTubeDB::TubeDB
`region`	vector of TubeDB region ids - see rTubeDB::query_regions (default NULL) Regions are used mainly for loading metadata from TubeDB localities.
`plot`	vector of localities ids see rTubeDB::query_region_plots rTubeDB::query_timeseries (default NULL) If plot is NULL, then all localities are loaded from whole region.
`sensor_ids`	list in format `list(tubedb_sensor_name=myClim_sensor_name)` (default NULL) If sensor names in TubeDB match the default sensor names in myClim, then the value is detected automatically.
`clean`	if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)
`silent`	if TRUE, then any information is not printed in console (default FALSE)
`aggregation`	parameter used in function rTubeDB::query_timeseries (default raw)
`quality`	parameter used in function rTubeDB::query_timeseries (default no)
`...`	other parameters from function rTubeDB::query_timeseries

Details

In case you store your microclimatic time-series in TubeDB, you can read data with TubeDB API into myClim object. You need to know database URL, username and password.

Value

myClim object in Raw-format

Examples

```
# Not run: To retrieve data from TubeDB, a running TubeDB server with a user account
#          and a secret password is required.
## Not run:
tubedb <- TubeDB(url="server", user="user", password="password")
data <- mc_read_tubedb(tubedb, region="ckras", plot=c("TP_KAR_19", "TP_KODA_61"))

## End(Not run)
```

`mc_read_wide` *Reading data from wide data.frame*

Description

This is universal function designed to read time-series and values from wide data.frame to myClim object. Useful for data not coming from supported microclimatic loggers. E.g. meteorological station data.

Usage

```
mc_read_wide(
  data_table,
  sensor_id = mc_const_SENSOR_real,
  sensor_name = NULL,
  clean = TRUE,
  silent = FALSE
)
```

Arguments

`data_table` data.frame with first column of POSIXct time format UTC timezone, followed by columns with (micro)climatic records. See details.

Columns:

- datetime column - POSIXct in UTC timezone is required
- Name of locality[1] - values
- ...
- Name of locality[n] - values

`sensor_id` define the sensor type, one of `names(mc_data_sensors)` (default `real`)

`sensor_name` custom name of sensor; if NULL (default) than `sensor_name` == `sensor_id`

`clean` if TRUE, then mc_prep_clean is called automatically while reading (default TRUE)

`silent` if TRUE, then any information is printed in console (default FALSE)

Details

The first column of input data.frame must be datetime column in POSIXct time format UTC timezone. Following columns represents localities. Column names are the localities names. All values in wide data.frame represents the same sensor type, e.g. air temperature. If you wish to read multiple sensors use mc_read_long or use mc_read_wide multiple times separately for each sensor type and that merge myClim objects with mc_prep_merge By default data are cleaned with function `mc_prep_clean()`. See function description. It detects holes in time-series, duplicated records or records in wrong order.

Value

myClim object in Raw-format

See Also

mc_read_long

`mc_reshape_long` *Export values to long table*

Description

This function converts myClim object to long R data.frame.

Usage

```
mc_reshape_long(data, localities = NULL, sensors = NULL, use_utc = TRUE)
```

Arguments

`data`	myClim object see myClim-package
`localities`	names of localities; if NULL then all (default NULL)
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`use_utc`	if FALSE, then the time shift from `tz_offset` metadata is used to correct (shift) the output time-series (default TRUE) In the Agg-format myClim object `use_utc = FALSE` is allowed only for steps shorter than one day. In myClim the day nd longer time steps are defined by the midnight, but this represent whole day, week, month, year... shifting daily, weekly, monthly... data (shift midnight) does not make sense in our opinion. But when user need more flexibility, then myClim Raw-format can be used, In Raw-format `use_utc` is not limited, user can shift an data without the restrictions. See myClim-package

Value

data.frame

columns:

- locality_id
- serial_number
- sensor_name
- height
- datetime
- time_to
- value

Examples

```
head(mc_reshape_long(mc_data_example_clean, c("A6W79", "A2E32"), c("TMS_T1", "TMS_T2")), 10)
```

mc_reshape_wide *Export values to wide table*

Description

This function converts myClim object to the R data.frame with values of sensor in wide format.

Usage

```
mc_reshape_wide(data, localities = NULL, sensors = NULL, use_utc = TRUE)
```

Arguments

`data`	myClim object see myClim-package
`localities`	names of localities; if NULL then all (default NULL)
`sensors`	names of sensors; if NULL then all (default NULL) see `names(mc_data_sensors)`
`use_utc`	if FALSE, then the time shift from `tz_offset` metadata is used to correct (shift) the output time-series (default TRUE) In the Agg-format myClim object `use_utc = FALSE` is allowed only for steps shorter than one day. In myClim the day nd longer time steps are defined by the midnight, but this represent whole day, week, month, year... shifting daily, weekly, monthly... data (shift midnight) does not make sense in our opinion. But when user need more flexibility, then myClim Raw-format can be used, In Raw-format `use_utc` is not limited, user can shift an data without the restrictions. See myClim-package

Details

First column of the output data.frame is datetime followed by the columns for every sensor. Name of the column is in format:

- localityid_loggerid_serialnumber_sensorname for Raw-format
- localityid_sensorname for Agg-format

The less complex wide table is returned when exporting single sensor ascross localities.

Value

data.frame with columns:

- datetime
- locality1_sensor1
- ...
- ...
- localityn_sensorn

Examples

```
example_tms_wideformat <- mc_reshape_wide(mc_data_example_raw, c("A6W79", "A2E32"),
                                          c("TMS_T1", "TMS_T2"))
```

---

mc_save *Save myClim object*

Description

This function was designed for saving the myClim data object to an .rds file, which can be later correctly loaded by any further version of myClim package with mc_load. This is the safest way how to store and share your myClim data.

Usage

```
mc_save(data, file)
```

Arguments

`data`	myClim object see myClim-package
`file`	path to output .rds file

Value

RDS file saved at the output path destination

Examples

```
tmp_dir <- tempdir()
tmp_file <- tempfile(tmpdir = tmp_dir)
mc_save(mc_data_example_agg, tmp_file)
file.remove(tmp_file)
```

---

mc_Sensor-class *Class for sensor definition*

Description

Sensor definitions are stored in mc_data_sensors.

Slots

`sensor_id` unique identifier of sensor (TMS_T1, TMS_T2, TMS_T3, TMS_moist, ...)

`logger` name of logger (TMS, Thermo, ...)

`physical` unit of sensor (T_C, moisture_raw, moisture, RH) (default NA)

`description` character info

`value_type` type of values (real, integer, logical) (default real)

`min_value` minimal value (default NA)

`max_value` maximal value (default NA)

`plot_color` color in plot (default "")

`plot_line_width` width of line in plot (default 1)

See Also

mc_data_sensors

`mc_SensorMetadata-class`

Class for sensor metadata

Description

Class for sensor metadata

Details

`sensor_id` must be one of the defined id in myClim. see mc_data_sensors. It is useful to select on of predefined, because it makes plotting and calculaton easier. Through `sensor_id` myClim assign pre-deined physicyl units or plotting colors see mc_Sensor.

Slots

`sensor_id` unique identifier of sensor (TMS_T1, TMS_T2, TMS_T3, TMS_moist, ...) mc_data_sensors e.g. TMS_T1, TMS_moist, snow_fresh...

`name` character, could be same as `sensor_id` but also defined by function or user.

`height` character

`calibrated` logical - detect if sensor is calibrated

See Also

myClim-package, mc_LoggerMetadata, mc_data_sensors

mc_states_delete *Delete sensor states (tags)*

Description

This function removes states (tags) defined by locality ID, sensor name, or tag value, or any combination of these three.

Usage

```
mc_states_delete(data, localities = NULL, sensors = NULL, tags = NULL)
```

Arguments

data	cleaned myClim object see myClim-package
localities	locality ids where delete states (tags). If NULL then all. (default NULL)
sensors	sensor names where delete states (tags). If NULL then all. (default NULL)
tags	specific tag to be deleted. If NULL then all. (default NULL)

Value

myClim object in the same format as input, with deleted sensor states

Examples

```
data <- mc_states_delete(mc_data_example_clean, localities="A1E05",
                         sensors=c(mc_const_SENSOR_Dendro_T, mc_const_SENSOR_Dendro_raw))
```

mc_states_from_sensor *Convert a sensor to a state*

Description

This function creates a new state from an existing logical (TRUE/FALSE) sensor and assigns this new state to selected existing sensors.

Usage

```
mc_states_from_sensor(
  data,
  source_sensor,
  tag,
  to_sensor,
  value = NA,
  inverse = FALSE
)
```

Arguments

`data`	myClim object see myClim-package
`source_sensor`	A logical sensor to be converted to states.
`tag`	A tag for the new states, e.g., "snow".
`to_sensor`	A vector of sensor names to which the new states should be attributed.
`value`	The value of the new states (default is NA)
`inverse`	A logical value. If FALSE, states are created for periods when `source_sensor` is TRUE (default is FALSE).

Details

The function is applicable only for logical (TRUE/FALSE) sensors. It allows you to convert such sensors into a state, represented as a tag. For example, you might calculate the estimation of snow cover using mc_calc_snow (TRUE/FALSE) and then want to remove temperature records when the logger was covered by snow. In this case, you can convert the snow sensor to a state, and then replace the values with NA for that state using mc_states_replace. In opposite case when you wish to keep e.g. only the moisture records when sensor was covered by snow, use `inverse = TRUE`.

Value

Returns a myClim object in the same format as the input, with added states.

Examples

```
data <- mc_calc_snow(mc_data_example_agg, "TMS_T2", output_sensor="snow")
data <- mc_states_from_sensor(data, source_sensor="snow", tag="snow", to_sensor="TMS_T2")
```

`mc_states_insert` *Insert new sensor states (tags)*

Description

This function inserts new states (tags) into the selected part of the sensor time-series. For more information about the structure of states (tags), see myClim-package. `mc_states_insert()` does not affect existing rows in the states (tags) table but only inserts new rows even if the new ones are identical with existing (resulting in duplicated states).

Usage

```
mc_states_insert(data, states_table)
```

Arguments

data
: cleaned myClim object see myClim-package

states_table
: Output of `mc_info_states()` can be used as template for input data.frame. data.frame with columns:

  - locality_id - the name of locality (in some cases identical to logger id, see mc_read_files)
  - logger_index - index of logger in myClim object at the locality. See mc_info_logger.
  - sensor_name - sensor name either original (e.g., TMS_T1, T_C), or calculated/renamed (e.g., "TMS_T1_max", "my_sensor01")
  - tag - category of state (e.g., "conflict", "error", "source", "quality")
  - start - start datetime
  - end - end datetime
  - value - value of tag (e.g., "out of soil", "c:/users/John/tmsData/data_911235678.csv")

Details

As a template for inserting states (tags), it is recommended to use the output of `mc_info_states()`, which will return the table with all necessary columns correctly named. The `sensor_name` and `value` columns are optional and do not need to be filled in.

When `locality_id` is provided but `sensor_name` is NA in the states (tags) table, states are inserted for all sensors within the locality.

The states (tags) are associated with the sensor time-series, specifically to the defined part of the time-series identified by start and end date times. A single time series can contain multiple states (tags) of identical or different types, and these states (tags) can overlap. Start and end date times are adjusted to fit within the range of logger/locality datetime and are rounded according to the logger's step. For instance, if a user attempts to insert a tag beyond the sensor time-series range, `mc_states_insert` will adjust the start and end times to fit the available measurements. If a user defines a start time as '2020-01-01 10:23:00' on a logger with a 15-minute step, it will be rounded to '2020-01-01 10:30:00'.

Value

myClim object in the same format as input, with inserted sensor states

Examples

```
states <- data.frame(locality_id="A1E05", logger_index=1, sensor_name="Thermo_T", tag="error",
                     start=lubridate::ymd_hm("2020-10-28 9:00"),
                     end=lubridate::ymd_hm("2020-10-28 9:30"))
data <- mc_states_insert(mc_data_example_clean, states)
```

mc_states_outlier *Create states for outlying values*

Description

This function creates a state (tag) for all values that are either above or below certain thresholds (min_value, max_value), or at break points where consecutive values of microclimate time-series suddenly jump down or up (positive_jump, negative_jump).

Usage

```
mc_states_outlier(
  data,
  table,
  period = NULL,
  range_tag = "range",
  jump_tag = "jump"
)
```

Arguments

data	myClim object see myClim-package
table	The table with outlying values (thresholds). You can use the output of mc_info_range(). The columns of the table are: • sensor_name - Name of the sensor (e.g., TMS_T1, TMS_moist, HOBO_T); see mc_data_sensors • min_value - Minimal value (threshold; all below are tagged) • max_value - Maximal value • positive_jump - Maximal acceptable increase between two consecutive values (next value is higher than the previous) • negative_jump - Maximal acceptable decrease between two consecutive values (next value is lower than the previous)
period	Period for standardizing the value of jump. If NULL, then the difference is not standardized (default NULL); see details. It is a character string usable by lubridate::period, for example, "1 hour", "30 minutes", "2 days".
range_tag	The tag for states indicating that the value is out of range (default "range").
jump_tag	The tag for states indicating that the difference between two consecutive values is too high (default "jump").

Details

The best way to use this function is to first generate a table (data.frame) with pre-defined minimum, maximum, and jump thresholds using the mc_info_range function. Then modify the thresholds as needed and apply the function (see example). All values above max_value and below min_value are tagged by default with the `range` tag. When consecutive values suddenly decrease by more than `negative_jump` or increase by more than `positive_jump`, such break points are tagged with the `jump` tag. It is possible to use only the `range` case, only the `jump` case, or both.

When the `period` parameter is used, the jump values are modified; range values are not affected. Depending on the logger step, the value of jump is multiplied or divided. For example, when the loggers are recording with a step of 15 minutes (900 s) and the user sets `period = "1 hour"` together with `positive_jump = 10`, then consecutive values differing by (10 * (15 / 60) = 2.5) would be tagged. In this example, but with recording step 2 hours (7200 s), consecutive values differing by (10 * (120 / 60) = 20) would be tagged.

Value

Returns a myClim object in the same format as the input, with added states.

Examples

```
range_table <- mc_info_range(mc_data_example_clean)
range_table$negative_jump[range_table$sensor_name == "TMS_moist"] <- 500
data <- mc_states_outlier(mc_data_example_clean, range_table)
```

`mc_states_replace` *Replace values by states with tag*

Description

This function replace values of sensors by states with tag.

Usage

```
mc_states_replace(data, tags, replace_value = NA)
```

Arguments

`data`	myClim object see myClim-package
`tags`	specific tag to be replaced.
`replace_value`	(default NA).

Value

myClim object in the same format as input, with replaced values

Examples

```
states <- data.frame(locality_id="A1E05", logger_index=1, sensor_name="Thermo_T", tag="error",
                     start=lubridate::ymd_hm("2020-10-28 9:00"),
                     end=lubridate::ymd_hm("2020-10-28 9:30"))
data <- mc_states_insert(mc_data_example_clean, states)
data <- mc_states_replace(data, "error")
```

`mc_states_to_sensor` *Convert states to logical (TRUE/FALSE) sensor*

Description

This function creates a logical (TRUE/FALSE) sensor from specified states.

Usage

```
mc_states_to_sensor(
  data,
  tag,
  to_sensor,
  source_sensor = NULL,
  inverse = FALSE
)
```

Arguments

`data`	myClim object see myClim-package
`tag`	The tag of states to be converted into a sensor.
`to_sensor`	A vector of names for the output sensors. `If `to_sensor` is a single sensor name, the logical sensor is created from the union of states across all sensors with the same tag. If `to_sensor` contains multiple sensor names, the length of the vector must match the length of `source_sensor`.`
`source_sensor`	A vector of sensors containing the states to be converted into a new sensor. If NULL, states from all sensors are used. (default is NULL)
`inverse`	A logical value. If TRUE, the sensor value is FALSE for state intervals (default is FALSE).

Details

The function allows you to create a TRUE/FALSE sensor based on a tag. By default, it generates a new sensor by combining all tags specified in the `tag` parameter from all available sensors at a particular logger or locality. If you specify a `source_sensor`, the function converts only the tags from that specific sensor. You can also create multiple new sensors from multiple tags by specifying more values in `to_sensor` and providing exactly the same number of corresponding values in `source_sensor`. For example, you can create one TRUE/FALSE sensor from states on a temperature sensor and another from tags on a moisture sensor.

If you use parameter `inverse = TRUE` you get FALSE for each record where tag is assigned to and FALSE for the records where tag is absent. By default you get TRUE for all the records where tag is assigned.

Value

Returns a myClim object in the same format as the input, with added sensors.

Examples

```
states <- data.frame(locality_id="A1E05", logger_index=1, sensor_name="Thermo_T", tag="error",
                     start=lubridate::ymd_hm("2020-10-28 9:00"),
                     end=lubridate::ymd_hm("2020-10-28 9:30"))
data <- mc_states_insert(mc_data_example_clean, states)
data <- mc_states_to_sensor(data, tag="error", to_sensor="error_sensor")
```

---

mc_states_update *Update sensor states (tags)*

Description

This function updates (replaces) existing states (tags). For more information about the structure of states (tags), see myClim-package. In contrast with mc_states_insert, which does not affect existing states (tags), mc_states_update deletes all old states and replaces them with new ones, even if the new states table contains fewer states than original object.

Usage

```
mc_states_update(data, states_table)
```

Arguments

`data`	cleaned myClim object see myClim-package
`states_table`	Output of `mc_info_states()` can be used as template for input data.frame. data.frame with columns: • locality_id - the name of locality (in some cases identical to logger id, see details of mc_read_files) • logger_index - index of logger in myClim object at the locality. See mc_info_logger. • sensor_name - sensor name either original (e.g., TMS_T1, T_C), or calculated/renamed (e.g., "TMS_T1_max", "my_sensor01") • tag - category of state (e.g., "conflict", "error", "source", "quality") • start - start datetime • end - end datetime • value - value of tag (e.g., "out of soil", "c:/users/John/tmsData/data_911235678.csv")

Details

As a template for updating states (tags), it is recommended to use the output of `mc_info_states()`, which will return the table with all necessary columns correctly named. The `sensor_name` and `value` columns are optional and do not need to be filled in.

The states (tags) are associated with the sensor time-series, specifically to the defined part of the time-series identified by start and end date times. A single time series can contain multiple states (tags) of identical or different types, and these states (tags) can overlap. Start and end date times are adjusted to fit within the range of logger/locality datetime and are rounded according to the logger's step. For instance, if a user attempts to insert a tag beyond the sensor time-series range, `mc_states_insert` will adjust the start and end times to fit the available measurements. If a user defines a start time as '2020-01-01 10:23:00' on a logger with a 15-minute step, it will be rounded to '2020-01-01 10:30:00'.

In contrast with mc_states_insert, the automatic filling of states when `locality_id` is provided but `sensor_name` is NA is not implemented in mc_states_update. When a user needs to update states (tags) for all sensors within the locality, each state (tag) needs to have a separate row in the input table.

Value

myClim object in the same format as input, with updated sensor states

Examples

```
states <- mc_info_states(mc_data_example_clean)
states$value <- basename(states$value)
data <- mc_states_update(mc_data_example_clean, states)
```

---

mc_TOMSTDataFormat-class

Class for reading TOMST logger files

Description

Provides the key for the column in source files. Where is the date, in what format is the date, in which columns are records of which sensors. The code defining the class is in section methods ./R/model.R

See Also

mc_DataFormat, mc_data_formats, mc_TOMSTJoinDataFormat

`mc_TOMSTJoinDataFormat-class`
Class for reading TMS join files

Description

Provides the key for the column in source files. Where is the date, in what format is the date, in which columns are records of which sensors. The code defining the class is in section methods ./R/model.R

Details

TMS join file format is the output of IBOT internal post-processing of TOMST logger files.

See Also

mc_DataFormat,mc_data_formats,mc_TOMSTDataFormat, mc_TOMSTJoinDataFormat

---

`myClimList` *Custom list for myClim object*

Description

Top level list for store myClim data. (see myClim-package) Rather service function used for checking, whether object is myClimList. The same time can be used to create standard R list from myClimList.

Usage

```
myClimList(metadata = NULL, localities = list())
```

Arguments

`metadata`	of data object
`localities`	list of licalities

Value

the list containing myClim object's metadata and localities

print.myClimList *Print function for myClim object*

Description

Function print metadata of myClim object and table from function mc_info().

Usage

```
## S3 method for class 'myClimList'
print(x, ...)
```

Arguments

x	myClim object see myClim-package
...	other parameters from function print for tibble tibble::tibble

Examples

```
print(mc_data_example_agg, n=10)
```

[.myClimList *Extract localities with []*

Description

Using [] for extract localities.

Usage

```
## S3 method for class 'myClimList'
x[...]
```

Arguments

x	myClim object see myClim-package
...	indexes for extract localities

Value

myClim object with subset of localities see myClim-package

Examples

```
filtered_data <- mc_data_example_raw[1:2]
```

Index