Package: rtorf (via r-universe)

June 7, 2026

Type Package

Title Observations, Footprints and Receptors ('rtorf') and HYSPLIT Configuration Tools

Version 4.3.2

Description Ingest, process, and visualize atmospheric observation data, specifically tailored for the NOAA Global Monitoring Laboratory (GML) ObsPack GLOBALView+ products <https://gml.noaa.gov/ccgg/obspack/index.html>. Includes specialized routines for filtering trace gas time-series data based on meteorological and quality control parameters. Additionally, streamlines the generation of configuration files for the Hybrid Single-Particle Lagrangian Integrated Trajectory (HYSPLIT) model <https://www.ready.noaa.gov/HYSPLIT.php>, supporting high-throughput parallel processing of trajectory simulations.

Depends R (>= 2.10)

Imports data.table, utils, lubridate, cptcity, graphics, ncdf4

SystemRequirements netcdf (>=4.0)

License MIT + file LICENSE

URL https://github.com/ibarraespinosa/rtorf, https://ibarraespinosa.github.io/rtorf/

BugReports https://github.com/ibarraespinosa/rtorf/issues/

Encoding UTF-8

RoxygenNote 7.3.3

Date 2026-03-30

Suggests knitr, rmarkdown, rslurm, testthat (>= 3.0.0)

VignetteBuilder knitr

NeedsCompilation yes

Config/testthat/edition 3

Config/pak/sysreqs libnetcdf-dev

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

Date/Publication 2026-05-08 21:39:19 UTC

RemoteUrl https://github.com/atmoschem/rtorf

RemoteRef HEAD

RemoteSha 6fd2206842c2fe138c67f6fdec646150f6e91aea

Contents

`fex` *File extension*

Description

file extension

Usage

```
fex(x)
```

Arguments

`x`	character vector giving file paths.

Value

A character vector of the same length as `x` containing the file extensions without the leading dot. Files without an extension return an empty string ("").

Note

source tools::file_ext

See Also

`tools::file_ext()`

Other helpers: `obs_footname()`, `obs_format()`, `obs_freq()`, `obs_list.dt()`, `obs_out()`, `obs_rbind()`, `obs_read_csvy()`, `obs_roundtime()`, `obs_trunc()`, `obs_write_csvy()`, `rtorf-deprecated`, `sr()`

Examples

```
## Not run:
# Do not run

## End(Not run)
```

get_nt *Check the max number of threads*

Description

get_nt check the number of threads in this machine

Usage

```
get_nt()
```

Value

Integer with the max number of threads

Examples

```
{
  get_nt()
}
```

obs_addltime *add local time*

Description

Calculate an approximation of local hour

Usage

```
obs_addltime(
  dt,
  timeUTC = "timeUTC",
  utc2lt = "site_utc2lst",
  longitude = "longitude",
  tz = "UTC",
  timeonly = FALSE
)
```

Arguments

dt	data.table
timeUTC	Character indicating the Time column as POSIXct
utc2lt	Character indicating the integer column to convert to local time if available
longitude	Character indicating the column with the lingitude
tz	Timezone, default "UTC"
timeonly	return only local_time column

Value

data.table with local time columns

Note

time depending n longitude is by John Miller (GML/NOAA)

See Also

Other time: `obs_addmtime()`, `obs_addstime()`, `obs_addtime()`

Examples

```
## Not run:
# Do not run

## End(Not run)
```

`obs_addmtime` *Add matlab time*

Description

This function add timeUTC as POSIX class, local time and ending sampling time

Usage

```
obs_addmtime(time, origin = "1970-01-01", tz = "UTC")
```

Arguments

`time`	numeric time from MATLAB
`origin`	default "1970-01-01"
`tz`	Timezone, default "UTC"

Value

return the same data.frame adding solar time

See Also

Other time: `obs_addltime()`, `obs_addstime()`, `obs_addtime()`

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <-  obs_summary(obs)
dt <- obs_read(index)
dt <- obs_addtime(dt)

## End(Not run)
```

---

obs_addstime *Add solar time*

Description

This function add timeUTC as POSIX class, local time and ending sampling time

Usage

```
obs_addstime(dt, tz = "UTC")
```

Arguments

dt	obspack data.table
tz	Timezone, default "UTC"

Value

return the same data.frame adding solar time

See Also

Other time: obs_addltime(), obs_addmtime(), obs_addtime()

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <-  obs_summary(obs)
dt <- obs_read(index)
dt <- obs_addtime(dt)

## End(Not run)
```

obs_addtime *Add times*

Description

This function add timeUTC as POSIX class, local time and ending sampling time

Usage

```
obs_addtime(dt, verbose = TRUE, tz = "UTC", timeonly = FALSE)
```

Arguments

dt	data.table
verbose	Logical, to show more info
tz	Timezone, default "UTC"
timeonly	return only timeUTC column

Value

A data.frame with with an index obspack.

Note

timeUTC is calculated based on the field column start_time, the timeUTC_end is calculated using this approach: 1. If the column time_interval is not found, proceed with the calculation using midpoint_time 2. Else, use column time_interval

See Also

Other time: obs_addltime(), obs_addmtime(), obs_addstime()

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <-  obs_summary(obs)
dt <- obs_read(index)
dt <- obs_addtime(dt)

## End(Not run)
```

obs_agg *Aggregates Observations by time*

Description

This function add aggregate cols by ‘key_time‘

Usage

```
obs_agg(
  dt,
 cols = c("year", "month", "day", "hour", "minute", "second", "time", "time_decimal",
  "value", "latitude", "longitude", "altitude", "pressure", "u", "v", "temperature",
    "type_altitude"),
 by = c("key_time", "site_code", "altitude_final", "type_altitude", "lab_1_abbr",
    "dataset_calibration_scale"),
  fn = "mean",
  verbose = TRUE
)
```

Arguments

dt	data.table
cols	Character which defines columns to be aggregated
by	String with columns to be aggregated
fn	Function to be applied to the columns, default mean
verbose	logical to show more information

Value

A data.frame with with an index.

Note

By default add column timeUTC based on the input column ‘key_time‘

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)

## End(Not run)
```

obs_convolve *obs_convolve*

Description

This function returns a arrays (or list of arrays) of convolved footpritns with flux

Usage

```
obs_convolve(
  foot_path = "AAA",
  name_foot = "foot1",
  flon = "foot1lon",
  flat = "foot1lat",
  time_foot,
  flux = "CTCO2",
  df_flux,
  flux_format = "%Y%m%d.nc",
  name_var_flux = 1,
  factor = 1e+09,
  fn = NULL,
  flip = FALSE,
  as_list = FALSE,
  verbose = TRUE
)
```

Arguments

foot_path	path for footprint (length 1).
name_foot	name of the footprint variable in the NetCDF file.
flon	name of lons
flat	name of lats
time_foot	Time of the footprints (in the name file) or third dimension
flux	String, default NOAA "CTCO2". Implies bio, ocn, fossil, fire.
df_flux	data.table with columns f (full path) and nf file name 'Ymd.nc'
flux_format	string with date format 'Ymd.nc'
name_var_flux	string or numeric of position for the name var in the flux format. e.g.: If missing, asusmes date ("2020-01-01"), if string something like ("flux"), if numeric, identify the var names and you select which of them (first, second, etc)
factor	number to multiply fluxes.
fn	string with function to aggregate convolved fluxes, e.g. 'mean', 'sum', 'max', etc.
flip	Logical, to vertically flip emissions, or not
as_list	Logical, to return list of arrays
verbose	Logical, to display more information

Value

An array of footprints and convolved footprints, or a list of footprints, convolved fooprints, latitudes, longitudes and time.

Note

main assumption is that fluxes have same spatial dimensions as footprints when flux is "monthly", "daily" or "yearly", it assumes flux variable has the same name as "

Examples

```
## Not run:
# do not run

## End(Not run)
```

`obs_find_receptors` *Compares expected receptors*

Description

This function creates a data.frame with the expected footprints in .nc and compare with the actual footprints. The idea is verify is there are missing footprints.

Usage

```
obs_find_receptors(
  path,
  year,
  month,
  day,
  hour,
  minute,
  lat,
  lon,
  alt,
  out = paste0(tempfile(), ".csvy"),
  verbose = FALSE
)
```

Arguments

`path`	String with path hwere are stored the footprints
`year`	numeric number
`month`	numeric number
`day`	numeric number

hour	numeric number
minute	numeric number
lat	numeric number
lon	numeric number
alt	numeric number
out	outfile path.
verbose	logical to show more information

Value

A data.frame with with expected footprints

Examples

```
## Not run:
# do not run
p <- "/path/to/continuous/"
# here we have year/month/hysplit*.nc
x <- dt
dt <- obs_find_receptors(p, year, month....)

## End(Not run)
```

---

obs_foot *obs_footprints*

Description

This function returns a arrays (or list of arrays) of convolved footpritns with flux

Usage

```
obs_foot(
  foot_path = "AAA",
  name_foot = "foot1",
  flon = "foot1lon",
  flat = "foot1lat",
  time_foot,
  fn = NULL,
  as_list = FALSE,
  verbose = TRUE
)
```

Arguments

foot_path	path for footprint (length 1).
name_foot	name of the footprint variable in the NetCDF file.
flon	name of lons
flat	name of lats
time_foot	Time of the footprints (in the name file) or third dimension
fn	string with function to aggregate convolved fluxes, e.g. 'mean', 'sum', 'max', etc.
as_list	Logical, to return list of arrays
verbose	Logical, to display more information

Value

An array of footprints, or a list of footprints, latitudes, longitudes and time.

Examples

```
## Not run:
# do not run

## End(Not run)
```

---

obs_foot_flip *obs_foot_flip*

Description

Reorders the axes of a 3-D footprint array produced by obs_traj_foot so that its dimension order matches the CF-convention expected by obs_nc / terra::rast(), or converts back in the opposite direction.

Usage

```
obs_foot_flip(arr, from = c("traj", "cf"), flip_lat = FALSE, flip_lon = FALSE)
```

Arguments

arr	A 3-D numeric array. Two layouts are accepted: • "traj" (default) — [lat, lon, time] as produced by obs_traj_foot. Converted to [lon, lat, time]. • "cf" — [lon, lat, time] (CF / terra layout). Converted back to [lat, lon, time].
from	Character string, either "traj" or "cf". Describes the current axis order of arr. Default "traj".

`flip_lat` Logical. If `TRUE` (default) the latitude axis is also reversed so that index 1 corresponds to the southern-most row, matching the south-to-north order expected by CF tools and `terra`. Set to `FALSE` if the array is already in south-to-north order.

`flip_lon` Logical. If `TRUE` (default) the longitude axis is reversed so that index 1 corresponds to the western-most column. Set to `FALSE` if the array is already in west-to-east order.

Details

Why this is needed. `obs_traj_foot` returns an array with dimensions `[lat, lon, time]` (R row-major: row = latitude index). NetCDF files written with `obs_nc` preserve that order, but `terra::rast()` follows the CF / GIS convention where the *first* spatial dimension is longitude (x-axis). Reading the file back with `terra` therefore transposes the spatial plane, which is why a `t()` call was needed on the raster.

Calling `obs_foot_flip(arr)` before passing the array to `obs_nc` (or after reading it back) resolves the mismatch without manual transposing.

Value

A 3-D array with reordered (and optionally flipped) axes.

See Also

`obs_traj_foot`, `obs_nc`

Examples

```
## Not run:
# foot.arr comes from obs_traj_foot: dims [lat, lon, time]
foot_cf <- obs_foot_flip(foot.arr)       # now [lon, lat, time]

obs_nc(
  lat      = lats,
  lon      = lons,
  time_nc  = time_vec,
  vars_out = "foot1",
  units_out = "(ppb/nanomol m-2 s-1)",
  nc_out   = "output.nc",
  larrays  = list(foot1 = foot_cf)
)
# terra::rast("output.nc") now reads correctly without t()

## End(Not run)
```

obs_footname *Expected footprint name*

Description

return the expected name for the NetCDF footprint

Usage

```
obs_footname(
  time = NULL,
  year,
  month,
  day,
  hour,
  minute,
  second,
  lat,
  lon,
  alt,
  fullpath = FALSE,
  out,
  ...
)
```

Arguments

time	POSIXct time to extract time variblaes
year	numeric number
month	numeric number
day	numeric number
hour	numeric number
minute	numeric number
second	numeric number, optional
lat	numeric number
lon	numeric number
alt	numeric number
fullpath	Logical, to add or not YYYY/MO/hysplit to id
out	outfile path.
...	data.table::fwrite arguments.

Value

A concatenation of characters representing a spatio-temporal point.

Note

source https://stackoverflow.com/a/47015304/2418532

IMPORTANT!!! # This function will generate the expected NetCDF file name. # It assumes that the name was generated under the following considerations: # time variables (year, month, day, etc) have a format of two digits, eg "0.1" # latitude and longitude have been round with 4 decimals # The format for latitude is 2 integers, a point and 4 decimals # The format for longitude is 3 integers, a point and 4 decimals

See Also

Other helpers: `fex()`, `obs_format()`, `obs_freq()`, `obs_list.dt()`, `obs_out()`, `obs_rbind()`, `obs_read_csvy()`, `obs_roundtime()`, `obs_trunc()`, `obs_write_csvy()`, `rtorf-deprecated`, `sr()`

Examples

```
## Not run:
# Do not run
obs_footname(time = Sys.time(),
             second = data.table::second(Sys.Time()),
             lat = 0,
             lon = 0,
             alt = 0)
obs_footname(year = 2020,
             month = 12,
             day = 30,
             hour = 9,
             minute = 54,
             lat = 3.2133,
             lon = 30.9131,
             alt = 497,
             fullpath = TRUE)
obs_footname(year = 2020,
             month = 12,
             day = 30,
             hour = 9,
             minute = 54,
             lat = 1,
             lon = -130.9131,
             alt = 497,
             fullpath = TRUE)

## End(Not run)
```

`obs_format` *Formatting data*

Description

Formatting data

Usage

```
obs_format()
```

Value

formatted characters

See Also

Other helpers: fex(), obs_footname(), obs_freq(), obs_list.dt(), obs_out(), obs_rbind(), obs_read_csvy(), obs_roundtime(), obs_trunc(), obs_write_csvy(), rtorf-deprecated, sr()

Examples

```
## Not run:
# do not run

## End(Not run)
```

---

obs_freq *return numeric vector in intervals*

Description

return numeric vector in intervals

Usage

```
obs_freq(x, freq, ...)
```

Arguments

x	numeric, longer than ‘freq‘
freq	numeric vector
...	findInterval arguments

Value

numeric vector

See Also

Other helpers: `fex()`, `obs_footname()`, `obs_format()`, `obs_list.dt()`, `obs_out()`, `obs_rbind()`, `obs_read_csvy()`, `obs_roundtime()`, `obs_trunc()`, `obs_write_csvy()`, `rtorf-deprecated`, `sr()`

Examples

```
## Not run:
# Do not run

## End(Not run)
```

`obs_grid` *obs_grid*

Description

add columns of

Usage

```
obs_grid(
  min.x,
  max.x,
  min.y,
  max.y,
  numpix.x,
  numpix.y,
  coarse.factor = 1,
  adaptive = TRUE
)
```

Arguments

`min.x`	grid indices
`max.x`	grid indices
`min.y`	grid indices
`max.y`	grid indices
`numpix.x`	number pixels x
`numpix.y`	number pixels y
`coarse.factor`	integer to coarse the grid resolution
`adaptive`	logical, to use adaptive gridding

Value

return footprint

See Also

Other helpers legacy: obs_id2pos(), obs_info2id(), obs_julian(), obs_normalize_dmass(), obs_traj_foot()

Examples

```
obs_grid(1, 10, 1, 10, 100, 100, 1)
```

---

obs_grid_check *Conservation check for gridded footprints*

Description

Compares the total foot between the outputs of obs_grid_simple and obs_grid_kernel and prints a brief summary to the console. A relative difference larger than 5% triggers a warning with diagnostic suggestions.

Usage

```
obs_grid_check(simple, kernel)
```

Arguments

simple	Output list from obs_grid_simple.
kernel	Output list from obs_grid_kernel.

Details

The kernel method loses a small amount of foot at the ±3 sigma boundary of the truncated Gaussian. Larger differences usually indicate a grid extent that is too tight (edge particles are discarded) or a bandwidth that is large relative to the domain.

Value

Invisibly returns a named numeric vector with three elements: simple (total foot from bin-and-sum), kernel (total foot from kernel method), and relative_diff_pct (absolute percentage difference relative to simple).

See Also

obs_grid_simple, obs_grid_kernel

Examples

```
## Not run:
bs <- obs_grid_simple(d, lon_min, lat_min, lon_max, lat_max)
gk <- obs_grid_kernel(d, lon_min, lat_min, lon_max, lat_max,
                      bandwidth = 0.2)
chk <- obs_grid_check(bs, gk)
chk["relative_diff_pct"]

## End(Not run)
```

`obs_grid_cube` *3D footprint gridding (lon, lat, time)*

Description

Bins particles into a 3D grid: longitude, latitude, and time. Supports two methods: "simple" (bin-and-sum) and "kernel" (Gaussian smoothing).

Usage

```
obs_grid_cube(
  x,
  lon_min,
  lat_min,
  lon_max,
  lat_max,
  res = 0.1,
  nt = 240L,
  t0 = 0,
  dt = 60,
  method = c("simple", "kernel"),
  bandwidth = res,
  use_haversine = FALSE,
  n_threads = 1L,
  npar = 1L
)
```

Arguments

`x`	A `data.table` or `data.frame` with columns `lat`, `lon`, `time` (minutes), and `foot`.
`lon_min`, `lat_min`, `lon_max`, `lat_max`	Grid extent (degrees).
`res`	Spatial resolution (degrees).
`nt`	Number of time layers in the output cube.
`t0`	Reference time for the first layer (minutes). Default `0`.

dt	Time step per layer (minutes). Default 60 (1 hour).
method	Character. Either "simple" for bin-and-sum or "kernel" for Gaussian smoothing. Default "simple".
bandwidth	Kernel standard deviation for method = "kernel". In degrees if use_haversine = FALSE, or metres if TRUE. Default equals res.
use_haversine	Logical. If TRUE, uses great-circle distances and bandwidth in metres for kernel smoothing.
n_threads	Number of OpenMP threads.
npar	Normalization factor (total particles released).

Value

A named list:

grid Numeric 3D array of dimension [ny x nx x nt].

lon, lat Cell-centre coordinates.

time Start time of each bin (minutes back from t0).

---

obs_grid_kernel *Gaussian kernel footprint gridding*

Description

Computes a gridded footprint by spreading each particle's foot value over neighbouring cells using a 2-D isotropic Gaussian kernel. The kernel integrates to 1 (cell area lon_res * lat_res is included in the weight), so the total foot is approximately conserved — the small residual arises from truncation at ±3 sigma.

Usage

```
obs_grid_kernel(
  x,
  lon_min,
  lat_min,
  lon_max,
  lat_max,
  lon_res = 0.1,
  lat_res = 0.1,
  bandwidth = lon_res,
  use_haversine = FALSE,
  n_threads = 1L,
  npar = 1L
)
```

Arguments

x	A data.table or data.frame with columns lat, lon, and foot (same units as obs_grid_simple).
lon_min	Western edge of the output grid (degrees).
lat_min	Southern edge of the output grid (degrees).
lon_max	Eastern edge of the output grid (degrees).
lat_max	Northern edge of the output grid (degrees).
lon_res	Cell width in longitude (degrees). Default 0.1.
lat_res	Cell height in latitude (degrees). Default 0.1.
bandwidth	Kernel standard deviation. Units depend on use_haversine: • use_haversine = FALSE (default) — degrees. Recommended starting value: lon_res (one cell). Increase to 2 * lon_res for sparser particle sets. • use_haversine = TRUE — metres. A typical value for 0.1-degree grids is 11000 (approximately one degree at mid-latitudes). Default lon_res.
use_haversine	Logical. If FALSE (default), Cartesian distance in degrees is used — fast and adequate for domains smaller than ~1000 km and latitudes below ~60 degrees. If TRUE, great-circle distance via the Haversine formula is used — accurate at high latitudes and continental-scale domains; bandwidth must then be supplied in metres. A warning is issued when use_haversine = TRUE and bandwidth < 100, which suggests degrees were passed instead of metres.
n_threads	Number of OpenMP threads. Default 1L.
npar	Number of particles for normalization. Default 1L.

Details

Use this function when particle density is too low to fill every grid cell, or for display purposes. For analyses that require exact conservation of the total sensitivity, prefer obs_grid_simple.

Value

A named list with the same structure as obs_grid_simple:

grid Numeric matrix [nlon x nlat] of kernel-smoothed foot.

lon Cell-centre longitude vector.

lat Cell-centre latitude vector.

Note

The Fortran routine skips particles with foot == 0 before entering the kernel loop, so sparse footprints with many zero-weight particles incur no unnecessary computation.

See Also

obs_grid_simple for the unsmoothed primary output, obs_grid_check to verify conservation between the two methods.

Examples

```
## Not run:
library(data.table)
dat <- fread("PARTICLE.DAT")
d   <- dat[time %in% sort(unique(time))[1:4]]

# Cartesian mode (degrees, fast)
gk <- obs_grid_kernel(
  x             = d,
  lon_min       = floor(min(d$lon))   - 0.1,
  lat_min       = floor(min(d$lat))   - 0.1,
  lon_max       = ceiling(max(d$lon)) + 0.1,
  lat_max       = ceiling(max(d$lat)) + 0.1,
  lon_res       = 0.1,
  lat_res       = 0.1,
  bandwidth     = 0.2,          # 2-cell sigma in degrees
  use_haversine = FALSE,
  n_threads     = 4L
)
image(gk$lon, gk$lat, log1p(gk$grid), main = "Gaussian kernel")

# Haversine mode (metres, accurate at high latitudes)
gk_hav <- obs_grid_kernel(
  x             = d,
  lon_min       = -130, lat_min = 55,
  lon_max       = -100, lat_max = 75,
  lon_res       = 0.1,  lat_res = 0.1,
  bandwidth     = 15000,        # 15 km
  use_haversine = TRUE,
  n_threads     = 4L
)

## End(Not run)
```

`obs_grid_simple` *Bin-and-sum footprint gridding*

Description

Bins STILT/HYSPLIT particles into a regular longitude/latitude grid and sums the `foot` sensitivity values per cell. This is the primary scientific output: no smoothing is applied, no foot is redistributed spatially, and the grid total exactly equals the particle total.

Usage

```
obs_grid_simple(
  x,
  lon_min,
```

```
  lat_min,
  lon_max,
  lat_max,
  res = 0.1,
  n_threads = 1L,
  npar = 1L
)
```

Arguments

x	A data.table or data.frame with at least the columns lat (degrees), lon (degrees), and foot (sensitivity, ppm / (umol m-2 s-1)).
lon_min	Western edge of the output grid (degrees).
lat_min	Southern edge of the output grid (degrees).
lon_max	Eastern edge of the output grid (degrees).
lat_max	Northern edge of the output grid (degrees).
res	Cell size in degrees. Both longitude and latitude use the same value (square cells). Default 0.1.
n_threads	Number of OpenMP threads passed to the Fortran routine. Default 1L. Increase for large particle sets (> ~50 000 rows); for smaller sets the thread-management overhead outweighs the gain.
npar	Number of particles used for normalization. The output grid is divided by this value to return the average footprint sensitivity. Default 1L.

Value

A named list with three elements:

grid Numeric matrix of dimension [nlon x nlat] containing the summed foot per cell, divided by npar. Row i corresponds to lon[i], column j to lat[j].

lon Numeric vector of cell-centre longitudes (length nlon).

lat Numeric vector of cell-centre latitudes (length nlat).

Note

Particles whose coordinates fall outside the rectangle [lon_min, lon_max) x [lat_min, lat_max) are silently discarded. Add a small buffer to the grid extent if edge particles matter.

See Also

obs_grid_kernel for a smoothed alternative, obs_grid_check to compare totals.

Examples

```
## Not run:
library(data.table)
dat <- fread("PARTICLE.DAT")
d   <- dat[time %in% sort(unique(time))[1:4]]   # first hour

bs <- obs_grid_simple(
  x         = d,
  lon_min   = floor(min(d$lon))   - 0.1,
  lat_min   = floor(min(d$lat))   - 0.1,
  lon_max   = ceiling(max(d$lon)) + 0.1,
  lat_max   = ceiling(max(d$lat)) + 0.1,
  res       = 0.1,
  n_threads = 4L
)
image(bs$lon, bs$lat, log1p(bs$grid), main = "Bin and sum")

## End(Not run)
```

`obs_hysplit_ascdata` *obs_hysplit_ascdata*

Description

This function creates a ASCDATA.CFG file for HYSPLIT model.

Usage

```
obs_hysplit_ascdata(
  llc = c(-90, -180),
  spacing = c(1, 1),
  n = c(180, 360),
  landusecat = 2,
  rough = 0.2,
  bdyfiles = "../bdyfiles/",
  ascdata = "ASCDATA.CFG"
)
```

Arguments

`llc`	Lower left corner, default c(-90.0, -180.0)
`spacing`	spacing in degress, default c(1.0, 1.0)
`n`	number of data points, default c(180, 360)
`landusecat`	land use category, default 2
`rough`	default roughness length (meters), default 0.2
`bdyfiles`	directory location of the data files, default '../bdyfiles/'
`ascdata`	file, default ASCDATA.CFG

Value

A ASCDATA.CFG file

Examples

```
{
# Do not run
ascdata_file <- tempfile()
obs_hysplit_ascdata(ascdata = ascdata_file)
cat(readLines(ascdata_file), sep =  "\n")
}
```

---

obs_hysplit_control *obs_hysplit_control*

Description

This function creates a CONTROL file for HYSPLIT model. It uses inputs from a data.frame with the receptor information.

Usage

```
obs_hysplit_control(
  df,
  year,
  month,
  day,
  hour,
  minute,
  lat,
  lon,
  alt,
  start_time = NULL,
  nlocations = 1,
  duration = -240,
  vertical_motion = 5,
  top_model_domain = 10000,
  met = c("hrrr", "nams", "gfs0p25"),
  nmet = abs(duration/24) + 1,
 metpath = c("/work/noaa/lpdm/metfiles/hrrr/", "/work/noaa/lpdm/metfiles/nams/",
    "/work/noaa/lpdm/metfiles/gfs0p25/"),
 metformat = c(hrrr = "%Y%m%d_hrrr", nams = "%Y%m%d_hysplit.t00z.namsa", gfs0p25 =
    "%Y%m%d_gfs0p25", era5 = "ERA5_%Y%m%d.ARL"),
  ngases = 1,
  gas = "Foot",
  emissions_rate = 0,
  hour_emissions = 0.01,
```

```
  release_start = NULL,
  nsim_grids = 1,
  center_conc_grids = c(0, 0),
  grid_spacing = c(0.1, 0.1),
  grid_span = c(30, 30),
  nconc = "cdump",
  nvert_levels = 1,
  height_vert_levels = 50,
  sampling_start_time = c(0, 0, 0, 0, 0),
  sampling_end_time = c(0, 0, 0, 0, 0),
  sampling_interval_type = c(0, abs(duration), 0),
  npol_depositing = 1,
  particle_params = c(0, 0, 0),
  dmwsrdre = c(0, 0, 0, 0, 0),
  wrhcicbc = c(0, 0, 0),
  radiactive_decay = 0,
  pol_res = 0,
  control = "CONTROL",
  verbose = FALSE
)
```

Arguments

df	data.frame of receptor information. Must include, "year", "month", "day", "hour" (0:23), "minute", "latitude", "longitude", "altitude"
year	year, if missing df.
month	month, if missing df.
day	day, if missing df.
hour	hour, if missing df.
minute	minute, if missing df.
lat	latitude, if missing df.
lon	longitude, if missing df.
alt	altitude, if missing df.
start_time	String to be added, default derived from df.
nlocations	number of locations.
duration	number of hours of release. (Negative is backwards in time).
vertical_motion	Vertical motion option. (0:data 1:isob 2:isen 3:dens 4:sigma 5:diverg 6:msl2agl 7:average 8:damped). . The default "data" selection will use the meteorological model's vertical velocity fields; other options include isobaric, isentropic, constant density, constant internal sigma coordinate, computed from the velocity divergence, a special transformation to correct the vertical velocities when mapped from quasi-horizontal surfaces (such as relative to MSL) to HYSPLIT's internal terrain following sigma coordinate, and a special option (7) to spatially average the vertical velocity. The averaging distance is automatically computed

from the ratio of the temporal frequency of the data to the horizontal grid resolution. Default 5

`top_model_domain`
: altitude above ground level (m).

`met`
: meteorological models to be used.

`nmet`
: Number of days for the meteorological files. Default is number of days in duration plus two days. nmet is the number of simultaneous input meteorological files. For instance, 11 means that for each meteorological grid, 11 files are expected. Usually, the files are daily. Note that the same number of files are required for each grid in this approach. Hysplit expects something like 2 11, which means 2 meteorological grids with 11 files each. The number 2 comes from the length of met files.

`metpath`
: paths for each meteorological model output.

`metformat`
: format to applied to the meteorological daily file

`ngases`
: Default 1.

`gas`
: default "Foot".

`emissions_rate`
: Default 0

`hour_emissions`
: hour release, depend of type of release, instantaneous 0.01, continuous 1.

`release_start`
: derived from df.

`nsim_grids`
: Number of simulated grids.

`center_conc_grids`
: center coordinates for conc grid.

`grid_spacing`
: grid spacing, default is 0.1 degree.

`grid_span`
: model extension in degrees by dimension.

`nconc`
: name of the concentration file, default is "cdump"

`nvert_levels`
: number of vertical levels

`height_vert_levels`
: hright vertical levels (50)

`sampling_start_time`
: 2 digits year, month, day, hour, minute

`sampling_end_time`
: 2 digits year, month, day, hour, minute

`sampling_interval_type`
: type, hour, minure

`npol_depositing`
: number of pollutant depositing

`particle_params`
: Particle diameter, density, and shape

`dmwsrdre`
: DepVel, MW, SurfReRa, DifRa, EHenry

`wrhcicbc`
: Wet removal: HC, incloud, belowcloud

`radiactive_decay`
: days

`pol_res`
: Pollutant Resuspension (1/m)

`control`
: name of the file, default "CONTROL"

`verbose`
: logical, to show more information

Value

A CONTROL file

Examples

```
{
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)
df <- dt[1]
control_file <- tempfile()
obs_hysplit_control(df, control = control_file)
ff <- readLines(control_file)

cat(ff, sep =  "\n")
}
```

`obs_hysplit_control_read` *obs_hysplit_control_read*

Description

This function creates a CONTROL file for HYSPLIT model. It uses inputs from a data.frame with the receptor information.

Usage

```
obs_hysplit_control_read(control = "CONTROL")
```

Arguments

`control`	CONTROL text file

Value

A list with CONTROL information

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)
df <- dt[1]
control_file <- tempfile()
obs_hysplit_control(df, control = control_file)
```

```
ff <- readLines(control_file)

cat(ff, sep =  "\n")
obs_hysplit_control_read(control_file)


## End(Not run)
```

---

obs_hysplit_setup *obs_hysplit_setup*

Description

This function creates a SETUP.CFG file for HYSPLIT model.

Usage

```
obs_hysplit_setup(
  idsp = 2,
  capemin = 500,
  vscales = -1,
  kbls = 1,
  kblt = 5,
  kmixd = 0,
  initd = 0,
  veght = 0.5,
  kmix0 = 150,
  numpar = 500,
  maxpar = 500,
  ichem = 8,
  krand = 4,
 varsiwant = c("time", "indx", "lati", "long", "zagl", "zsfc", "foot", "samt", "temp",
    "dswf", "mlht", "dens", "dmas", "sigw", "tlgr"),
  ivmax = length(varsiwant),
  outdt = 15,
  extra_params,
  bypass_params,
  setup = "SETUP.CFG"
)
```

Arguments

idsp	particle dispersion scheme 1:HYSPLIT 2:STILT
capemin	-1 no convection; -2 Grell convection scheme; -3 extreme convection; >0 enhanced vertical mixing when CAPE exceeds this value (J/kg)
vscales	vertical Lagrangian time scale (sec) for stable PBL
kbls	boundary layer stability derived from 1:fluxes 2:wind_temperature

kblt	boundary layer turbulence parameterizations 1:Beljaars 2:Kanthar 3:TKE 4:Measured 5:Hanna
kmixd	mixed layer obtained from 0:input 1:temperature 2:TKE 3:modified Richardson
initd	initial distribution, particle, puff, or combination
veght	Height below which particle's time is spent is tallied to calculate footprint for PARTICLE_STILT.DAT less than or equal to 1. 0: fraction of PBL height; greater than 1.0: heightAGL (m)
kmix0	minimum mixing depth
numpar	number of puffs or particles to released per cycle
maxpar	maximum number of particles carried in simulation
ichem	chemistry conversion modules 0:none 1:matrix 2:convert 3:dust 4: conc grid equal to met grid, 5: divide output mass by air density (kg/m3) to sum as mixing ration, 7: transport deposited particles on the ocean surface, 8: stilt mode mixing ratio and varying layer, 9: set concentration layer one to a fraction of the boundary layer, 10: restructure concentration grid into time-varying transfer matrix, 11: enable daughter produyct calculation.
krand	0 method to calculate random number 0=precompute if NUMPAR greater than 5000 or dynamic if NUMPAR less than or equal to 5000; 1=precalculated; 2=calculated in pardsp; 3=none; 4=random initial seed number and calculated in pardsp; 10=same as 0 with random initial seed for non-dispersion applications; 11=same as 1 with random initial seed for non-dispersion applications; 12=same as 2 with random initial seed for non-dispersion applications; 13=same as 3 with random initial seed for non-dispersion applications
varsiwant	='TIME','INDX','LONG','LATI','ZAGL','SIGW','TLGR','ZSFC', TEMP', 'SAMT','FOOT','SHTF',' variables written to PARTICLE_STILT.DAT. However, the default in this case is: c('time','indx', 'lati','long','zagl', 'zsfc','foot','samt', 'temp', 'dswf','mlht','dens','dmas','sigw','tlgr'
ivmax	0 number of variables written to PARTICLE_STILT.DAT. Must equal the number of variables listed for variable VARSIWANT
outdt	Default 15. defines the output frequency in minutes of the endpoint positions in the PARTICLE.DAT file when the STILT emulation mode is configured. The default value of 0 results in output each time step while the positive value gives the output frequency in minutes. A negative value disables output. The output frequency should be an even multiple of the time step and be evenly divisible into 60. In STILT emulation mode, the time step is forced to one minute.
extra_params	more parameters
bypass_params	named vector of characters to bypass all other arguments. If this list is available, only the content of this list will be used to write SETUP.CFG file and not other arguments.
setup	Default SETUP.CFG

Value

A SETUP.CFG file

Note

The var description comes from hysplit 5.3 manual page 214 From the hysplit user guide: STILT mode The STILT model incorporates the variation of HYSPLIT developed by Lin et al. (2003 - JGR, VOL. 108, NO. D16, 4493, doi:10.1029/2002JD003161) that can be used to estimate upwind surface fluxes from atmospheric measurements. Two changes are introduced; the mass summation is divided by air density resulting in a mixing ratio output field (ICHEM=6) and the lowest concentration summation layer (concentration layer top depth) is permitted to vary with the mixed layer depth (ICHEM=9). The ICHEM=8 switch turns on both density and varying layer depth. Two text files of particle position information (PARTICLE.DAT and PARTICLE_STILT.DAT) at each time step will also be created unless the namelist parameter OUTDT defining the output interval (min) is changed. PARTICLE_STILT.DAT follows the same format as STILT. The footprint output in PARTICLE.DAT represents particles that were below 50 particles that were below a user defined height (VEGHT).

Examples

```
{
# Do not run
# default
setup_file <- tempfile()
obs_hysplit_setup(setup = setup_file)
cat(readLines(setup_file),sep =  "\n")
# bypass
setup_file <- tempfile()
obs_hysplit_setup(bypass_params = c(lala = 1), setup = setup_file)
cat(readLines(setup_file),sep =  "\n")
}
```

`obs_id2pos` *obs_id2pos*

Description

return ndata.frame based on footprint/receptor id

Usage

```
obs_id2pos(id, sep = "x", asdf = FALSE)
```

Arguments

`id`	string
`sep`	string
`asdf`	Logical, to return as data.frame or not

Details

Helpers Legacy

Value

data.frame with time and location based on input

See Also

Other helpers legacy: obs_grid(), obs_info2id(), obs_julian(), obs_normalize_dmass(), obs_traj_foot()

Examples

```
## Not run:
# Do not run
id <- '2002x08x03x10x45.00Nx090.00Ex00030'
(obs_id2pos(id, asdf = TRUE) -> dx)
id <- c('2002x08x03x10x00x45.000Nx090.000Ex00030',
        '2002x08x03x10x55x45.335Sx179.884Wx00030')
(obs_id2pos(id) -> dx)
(obs_id2pos(id, asdf = TRUE) -> dx)
(obs_id2pos(rep(id, 2)) -> dx)
(obs_id2pos(rep(id, 2), asdf = TRUE) -> dx)

## End(Not run)
```

---

obs_info2id *obs_info2id*

Description

return footprint/receptor id

Usage

```
obs_info2id(yr, mo, dy, hr, mn = 0, lat, lon, alt, sep = "x", long = TRUE)
```

Arguments

yr	year
mo	month
dy	day
hr	hour
mn	minute
lat	latitude
lon	longitude
alt	altitude above ground level
sep	character, default"x"
long	Logical, to add minute, and rounded with 2 decimals, instead of 4. default TRUE

Value

a string with the receptor id

See Also

Other helpers legacy: obs_grid(), obs_id2pos(), obs_julian(), obs_normalize_dmass(), obs_traj_foot()

Examples

```
## Not run:
# Do not run
obs_info2id(yr = 2002,
            mo = 8,
            dy = 3,
            hr = 10,
            mn = 0,
            lat = 42,
            lon = -90,
            alt = 1) [1]

## End(Not run)
```

---

obs_julian *obs_julian*

Description

return numeric

return footprint/receptor id

Usage

```
obs_julian(y, m, d, origin., legacy = FALSE, verbose = TRUE)

obs_julian(y, m, d, origin., legacy = FALSE, verbose = TRUE)
```

Arguments

y	year
m	month
d	day
origin.	string
legacy	logical, to use legacy code
verbose	logical, to show more messages

Details

Helpers Legacy

Value

returns day since 1/1/1960

a string with the receptor id

See Also

Other helpers legacy: obs_grid(), obs_id2pos(), obs_info2id(), obs_normalize_dmass(), obs_traj_foot()

Other helpers legacy: obs_grid(), obs_id2pos(), obs_info2id(), obs_normalize_dmass(), obs_traj_foot()

Examples

```
## Not run:
# Do not run
obs_julian(1, 2020, 1)

## End(Not run)
## Not run:
# Do not run
obs_julian(y = 2002,
            m = 8,
            d = 3,
            legacy = TRUE)
obs_julian(y = 2002,
            m = 8,
            d = 3,
            legacy = FALSE)

## End(Not run)
```

`obs_list.dt` *list.dt*

Description

Some treatments for list of data.frames

Usage

```
obs_list.dt(ldf, na, verbose = TRUE)
```

Arguments

ldf	list of data.frames
na	common names in the final data.frame
verbose	Logical to show more information

Value

long data.table

Note

1. Filter out empty data.frames 2. identify common names 3. rbindlist and return data.table

See Also

Other helpers: fex(), obs_footname(), obs_format(), obs_freq(), obs_out(), obs_rbind(), obs_read_csvy(), obs_roundtime(), obs_trunc(), obs_write_csvy(), rtorf-deprecated, sr()

Examples

```
## Not run:
# Do not run

## End(Not run)
```

---

obs_meta *Read obspack metadata*

Description

Read obspack metadata

Usage

```
obs_meta(
  index,
  verbose = TRUE,
  n_site_code = 15,
  n_site_name = 15,
  n_site_latitude = 18,
  n_site_longitude = 19,
  n_site_country = 18,
  n_dataset_project = 21,
  n_lab = 16,
  n_scales = 31,
  n_site_elevation = 20,
```

```
  n_altitude_comment = 22,
  n_utc = 18,
  fill_value = -1e+34,
  as_list = FALSE
)
```

Arguments

index	data.table
verbose	Logical to show more information
n_site_code	number of characters extraced from metadata after search
n_site_name	number of characters extraced from metadata after search
n_site_latitude	number of characters extraced from metadata after search
n_site_longitude	number of characters extraced from metadata after search
n_site_country	number of characters extraced from metadata after search
n_dataset_project	number of characters extraced from metadata after search
n_lab	number of characters extraced from metadata after search
n_scales	number of characters extraced from metadata after search
n_site_elevation	number of characters extraced from metadata after search
n_altitude_comment	number of characters extraced from metadata after search
n_utc	number of characters extraced from metadata after search
fill_value	fill value. Appeared in aoa_aircraft-flask_19_allvalid.txt
as_list	Logical to return as list

Value

A data.frame with with an index obspack.

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_meta(index)

## End(Not run)
```

obs_nc *obs_nc*

Description

Creates NetCDF based on dimension from another NetCDF with custom dimensions, and attributes

Usage

```
obs_nc(
  lat,
  lon,
  time_nc,
  vars_out = c("total", "bio", "ocn", "fossil", "fire"),
  units_out,
  nc_out,
  larrays,
  verbose = FALSE
)
```

Arguments

lat	lats array
lon	longs arrau
time_nc	Times.
vars_out	names for the variables to be created in the NetCDF.
units_out	units for the NetCDF variables to be created. NO DEFAULT.
nc_out	path for the created NetCDF.
larrays	list of arrays, length equal to vars_out.
verbose	Logical, to display more information.

Value

A NetCDF file with the custom attributes and units

Examples

```
# nc_path <- paste0("Z:/footprints/aircraft/flask/2018",
# "/04/hysplit2018x04x08x15x15x38.7459Nx077.5584Wx00594.nc")
# foot <- obs_nc_get(nc_path, all = TRUE)
# nco <- paste0(tempfile(), "2.nc")
# file.remove(nco)
# obs_nc(lat = foot$lat,
#        lon = foot$lon,
#        time_nc = ISOdatetime(2018, 4, 8, 15, 15, 38),
#        units_out = "(ppb/nanomol m-2 s-1)*nanomol m-2 s-1",
```

```
#        vars_out = c("a", "b"),
#        nc_out  = nco,
#        larrays = list(a = foot, b = foot),
#        verbose = TRUE)
```

---

obs_nc_get *obs_nc_get*

Description

Reads NetCDF var and returns the spatial array

Usage

```
obs_nc_get(
  nc_path = "AAA",
  nc_name = "foot1",
  nc_lat = "foot1lat",
  nc_lon = "foot1lon",
  verbose = FALSE,
  all = FALSE
)
```

Arguments

nc_path	String pointing to the target NetCDF
nc_name	String indicating the spatial array
nc_lat	String to extract latitude.
nc_lon	String to extract longitude
verbose	Logical, to display more information.
all	Logical, if TRUE, return list of array, lon and lats

Value

Array of convolved footprints, or lis tof convolved fluxes and lat lon

Examples

```
#nc_path <- paste0("Z:/footprints/aircraft/flask/2018/04",
#"/hysplit2018x04x08x15x15x38.7459Nx077.5584Wx00594.nc")
#f <- obs_nc_get(nc_path = nc_path)
```

obs_normalize_dmass *obs_normalize_dmass*

Description

add columns of

Usage

```
obs_normalize_dmass(part = NULL)
```

Arguments

part	particle data.table

Value

return footprint

See Also

Other helpers legacy: obs_grid(), obs_id2pos(), obs_info2id(), obs_julian(), obs_traj_foot()

Examples

```
## Not run:
# Do not run

## End(Not run)
#' # Do not run
```

obs_out *outersect*

Description

Just the opposite of intersect

Usage

```
obs_out(x, y)
```

Arguments

x	vector
y	vector

Details

Helpers

Value

vector opposite of intersect

See Also

Other helpers: fex(), obs_footname(), obs_format(), obs_freq(), obs_list.dt(), obs_rbind(), obs_read_csvy(), obs_roundtime(), obs_trunc(), obs_write_csvy(), rtorf-deprecated, sr()

Examples

```
## Not run:
#do not run

## End(Not run)
```

---

obs_plot *Read obspack metadata*

Description

Read obspack metadata

Usage

```
obs_plot(
  dt,
  time,
  y = "value",
  yfactor = 1,
  colu = "site_code",
  type = "p",
  n = if (length(unique(dt[[colu]])) == 1) unique(dt[[colu]]) else
    unique(dt[[colu]])[1:2],
  pal = cptcity::find_cpt("qual")[6],
  verbose = TRUE,
  xlab = "time",
  ylab = "value",
  xlim = range(dt[[time]], na.rm = TRUE),
  ylim = range(dt[[y]], na.rm = TRUE),
  ...
)
```

Arguments

dt	data.table
time	x axis column (time)
y	y axis column, default "value"
yfactor	factor of y
colu	column to plot by color, default site_code
type	type of plot, default "p"
n	Character indicating ‘colu‘ to subset, for instance, if you want to plot "site_code", here include all the site_code that you want to plot
pal	Color palette name see cpt, default "cb_qual_Accent_08"
verbose	Logical to show more information
xlab	Character, xlab
ylab	Character, ylab
xlim	x limits
ylim	y limits
...	plot arguments

Value

Plot

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)
obs_plot(dt, time = "time")

## End(Not run)
```

---

obs_rbind *rbind obspack*

Description

return rbind obs data.tables

Usage

```
obs_rbind(dt1, dt2, verbose = TRUE)
```

Arguments

dt1	first data.table
dt2	second data.table
verbose	logical to show more information

Value

numeric vector

See Also

Other helpers: fex(), obs_footname(), obs_format(), obs_freq(), obs_list.dt(), obs_out(), obs_read_csvy(), obs_roundtime(), obs_trunc(), obs_write_csvy(), rtorf-deprecated, sr()

Examples

```
## Not run:
# Do not run

## End(Not run)
```

---

obs_read *Read obspack (.txt)*

Description

Each obspack file has a header with metadata and this function reads selected fields from the metadata and add them as columns. This new columns are used later to be filtered

Usage

```
obs_read(
  index,
  categories = "flask",
  expr = NULL,
  verbose = TRUE,
  n_site_code = 15,
  n_site_latitude = 18,
  n_site_longitude = 19,
  n_site_name = 15,
  n_site_country = 18,
  n_dataset_project = 21,
  n_lab = 16,
  n_scales = 31,
  n_site_elevation = 20,
  n_altitude_comment = 22,
```

```
  n_utc = 18,
  fill_value = -1e+34,
  as_list = FALSE
)
```

Arguments

index	data.table
categories	character; ONE category : of c("aircraft-pfp", "aircraft-insitu", "surface-insitu", "tower-insitu", "aircore", "surface-pfp", "shipboard-insitu", "flask").
expr	String expressions to filter data.table internally
verbose	Logical to show more information
n_site_code	number of characters extratced from metadata after search
n_site_latitude	number of characters extracted from metadata after search
n_site_longitude	number of characters extracted from metadata after search
n_site_name	number of characters extracted from metadata after search
n_site_country	number of characters extracted from metadata after search
n_dataset_project	number of characters extracted from metadata after search
n_lab	number of characters extracted from metadata after search
n_scales	number of characters extracted from metadata after search
n_site_elevation	number of characters extracted from metadata after search
n_altitude_comment	number of characters extracted from metadata after search
n_utc	number of characters extracted from metadata after search
fill_value	fill value. Appeared in aoa_aircraft-flask_19_allvalid.txt
as_list	Logical to return as list

Value

A data.frame with with an index obspack.

Note

The identification of the altitude and type is critical. The approach used here consists of: 1. Identify agl from the name of the tile. 2. If magl not present, search dill_values used in elevation and transform them into NA (not available) 3. If magl is not present, agl = altitude - elevation 4. If there are some NA in elevation, will result some NA in agl 5. A new column is added named ‘altitude_final‘ to store agl or asl 6. Another column named ‘type_altitude‘ is added to identify "magl" or "masl" 7. If there is any case NA in ‘altitude_final‘, ‘type_altitude‘ is "not available"

Then, the relationship with hysplit is:

type_altitude	hysplit
magl	agl
masl	asl
not available	f-PBL

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)
obs_read(index, expr = "altitude_final == '5800'")

## End(Not run)
```

`obs_read_csvy` *reads CSVY*

Description

Reads CSVY, print YAML and fread file.

Usage

```
obs_read_csvy(f, n = 100, ...)
```

Arguments

`f`	path to csvy file
`n`	number of files to search for "—" yaml
`...`	extra data.table arguments

Value

a data.table from a csvy file

See Also

Other helpers: `fex()`, `obs_footname()`, `obs_format()`, `obs_freq()`, `obs_list.dt()`, `obs_out()`, `obs_rbind()`, `obs_roundtime()`, `obs_trunc()`, `obs_write_csvy()`, `rtorf-deprecated`, `sr()`

Examples

```
## Not run:
# Do not run
df <- data.frame(a = rnorm(n = 10),
                 time = Sys.time() + 1:10)

f <- paste0(tempfile(), ".csvy")
notes <- c("notes",
           "more notes")
obs_write_csvy(dt = df, notes = notes, out = f)
s <- obs_read_csvy(f)
s
# or
readLines(f)
data.table::fread(f)

## End(Not run)
```

---

obs_read_nc *Read obspack (.nc)*

Description

Each obspack file has a header with metadata and this function reads selected fields from the metadata and add them as columns. This new columns are used later to be filtered

Usage

```
obs_read_nc(
  index,
  categories = "flask",
  att = FALSE,
  expr = NULL,
  solar_time = if (grepl("aircraft", categories)) FALSE else TRUE,
  as_list = FALSE,
  verbose = FALSE,
  warnings = FALSE
)
```

Arguments

index	data.table
categories	character; ONE category : of c("aircraft-pfp", "aircraft-insitu", "surface-insitu", "tower-insitu", "aircore", "surface-pfp", "shipboard-insitu", "flask").
att	Logical, if global attributes should be added to data.table
expr	String expressions to filter data.table internally

solar_time	Logical, add solar time? Default: if categories include aircraft, FALSE, otherwise, TRUE
as_list	Logical to return as list
verbose	Logical to show more information
warnings	Logical to show warnings when reading NetCDF, especially global attributes

Value

A data.frame with with an index obspack.

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)

## End(Not run)
```

---

obs_read_nc_att *Read obspack attributes (.nc)*

Description

Each obspack file has a header with metadata and this function reads selected fields from the metadata and add them as columns. This new columns are used later to be filtered

Usage

```
obs_read_nc_att(index, as_list = FALSE, verbose = FALSE, warnings = FALSE)
```

Arguments

index	data.table
as_list	Logical to return as list
verbose	Logical to show more information
warnings	Logical to show warnings when reading NetCDF, especially global attributes

Value

A data.frame with with an index obspack.

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)

## End(Not run)
```

`obs_roundtime` *round seconds from "POSIXct" "POSIXt" classes*

Description

return rounded seconds from time

Usage

```
obs_roundtime(x, n = 10)
```

Arguments

x	time as "POSIXct" "POSIXt"
n	factor

Value

numeric vector

See Also

Other helpers: `fex()`, `obs_footname()`, `obs_format()`, `obs_freq()`, `obs_list.dt()`, `obs_out()`, `obs_rbind()`, `obs_read_csvy()`, `obs_trunc()`, `obs_write_csvy()`, `rtorf-deprecated`, `sr()`

Examples

```
## Not run:
# Do not run
x <- Sys.time() + seq(1, 55, 1)
paste0(x,"  ",
       obs_roundtime(x), "  ",
       obs_freq(data.table::second(x),
                seq(0, 55, 10)))

## End(Not run)
```

obs_select_sec *Select Observations by closest time (seconds)*

Description

This function select by closest time. Good for aircrafts

Usage

```
obs_select_sec(
  dt,
  origin = "1970-01-01",
  tz = "UTC",
  seconds = 30,
  verbose = TRUE
)
```

Arguments

dt	data.table
origin	a date-time object, default "1970-01-01"
tz	time zone string
seconds	seocnds target to select and filter data
verbose	logical to show more information

Value

A filtered data.frame.

Note

By default add column timeUTC based on the input column ‘key_time‘

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)
dx <- obs_read(index, expr = "altitude_final == '5800'")
dx <- obs_addtime(dx[, -"site_code"])
dy <- obs_select_sec(dx)

## End(Not run)
```

obs_summary *Summary of the ObsPack files (.txt)*

Description

This function returns a data.frame index of all files available in ObsPack.

This function returns a data.frame index of all files available in obspack.

Usage

```
obs_summary(
  obs,
 categories = c("aircraft-pfp", "aircraft-insitu", "surface-insitu", "tower-insitu",
    "aircore", "surface-pfp", "shipboard-insitu", "flask"),
  lnchar = 11,
  out = paste0(tempfile(), "_index.csv"),
  verbose = TRUE,
  aslist = FALSE
)

obs_index(
  obs,
 categories = c("aircraft-pfp", "aircraft-insitu", "surface-insitu", "tower-insitu",
    "aircore", "surface-pfp", "shipboard-insitu", "flask"),
  lnchar = 11,
  out = paste0(tempfile(), "_index.csv"),
  verbose = TRUE
)
```

Arguments

obs	Path to the Obspack GLOBALview txt data
categories	character; default are c("aircraft-pfp", "aircraft-insitu", "surface-insitu", "tower-insitu", "aircore", "surface-pfp", "shipboard-insitu", "flask"). The ideia is that,as the file names include these words, this function identifies which files has these words and add them as columns.
lnchar	Integer; last nchar, default = 11.
out	Path to the Obspack index output
verbose	Logical to show more information
aslist	Logical to return list of index and summary

Value

A data.frame with with an index of the obspack Globalview.

A data.frame with with an index of the obspack Globalview.

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)

## End(Not run)
{
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)

## End(Not run)
}
```

`obs_table` *Obspack Table Summary*

Description

This function reads the obsPack directory providing a summary for the columns: "value", "time", "time_decimal", "latitude" and "longitude". These summary are made by the columns "name", "sector", "site_name", "site_country", "type_altitude", "lab_1_abbr" and "site_utc2lst"

Usage

```
obs_table(
  df,
  cols = c("value", "time", "time_decimal", "latitude", "longitude")
)
```

Arguments

`df`	data.table
`cols`	String of columns to be summarized.

Value

A data.frame with with an index obspack.

Examples

```
## Not run:
# Do not run
obs <- system.file("data-raw", package = "rtorf")
index <- obs_summary(obs)
dt <- obs_read(index)
```

```
dx <- obs_table(dt)

## End(Not run)
```

---

obs_traj_foot *obs_traj_foot*

Description

return trajectory

Usage

```
obs_traj_foot(
  ident,
  part = NULL,
  timelabel = NULL,
  pathname = "",
  foottimes = 0:240,
  zlim = c(0, 0),
  coarse = 1,
  dmassTF = TRUE,
  numpix.x = 70,
  numpix.y = 70,
  lon.ll = -145,
  lat.ll = 11,
  lon.res = 1,
  lat.res = 1,
  npar = 500,
  eps.global = 0.1,
  adaptive = TRUE,
  center = FALSE,
  terminal_background = TRUE
)
```

Arguments

ident	foot id
part	data.table with PARTICLE.DAT information
timelabel	timelabel
pathname	pathname
foottimes	vector of times between which footprint or influence will be integrated
zlim	zlim
coarse	default 1
dmassTF	default TRUE weighting by accumulated mass due to violation of mass conservation in met fields

numpix.x	number of pixels in x directions in grid
numpix.y	number of pixels in y directions in grid
lon.ll	lower left lon
lat.ll	lower left lat
lon.res	resolution
lat.res	resolution
npar	default 500, number of particles hysplit was run with (required in order to account for those cases where a thinned particle table that may not contain all particle indices is used)
eps.global	default 0.01
adaptive	logical, if TRUE (default), uses adaptive gridding resolution in time.
center	logical, if TRUE (default FALSE), lon.ll and lat.ll are treated as cell centers and adjusted to edges internally.
terminal_background	logical, if TRUE (default), particles entering the background area are removed from the simulation from that point onward.

Value

return footprint

See Also

Other helpers legacy: obs_grid(), obs_id2pos(), obs_info2id(), obs_julian(), obs_normalize_dmass()

Examples

```
## Not run:
# Do not run

## End(Not run)
```

---

obs_trunc *Trunc numbers with a desired number of decimals*

Description

Trunc numbers with a specified number of decimals.

Usage

```
obs_trunc(n, dec)
```

Arguments

n	Numeric number
dec	Integer, number of decimals

Value

truncated number

Note

source https://stackoverflow.com/a/47015304/2418532

See Also

Other helpers: fex(), obs_footname(), obs_format(), obs_freq(), obs_list.dt(), obs_out(), obs_rbind(), obs_read_csvy(), obs_roundtime(), obs_write_csvy(), rtorf-deprecated, sr()

Examples

```
## Not run:
# Do not run
# in bash:
# printf "%07.4f" 72.05785
# results in 72.0578
# but:
formatC(72.05785, digits = 4, width = 8, format = "f", flag = "0")
# results in
"072.0579"
# the goal is to obtain the same trunc number as using bash, then:
formatC(obs_trunc(72.05785, 4),
        digits = 4,
        width = 8,
        format = "f",
        flag = "0")

## End(Not run)
```

---

obs_write_csvy *Generates YAML and write data.frame*

Description

Add YAML header info and writes a the data.frame into disk. The YAML section includes notes and str(dt).

Usage

```
obs_write_csvy(
  dt,
  notes,
  out = paste0(tempfile(), ".csvy"),
  sep = ",",
  nchar.max = 80,
  ...
)
```

Arguments

dt	data.table
notes	notes.
out	outfile path.
sep	The separator between columns. Default is ",".
nchar.max	Max nchar for str.
...	extra data.table arguments

Value

a csvy file to disk

See Also

Other helpers: fex(), obs_footname(), obs_format(), obs_freq(), obs_list.dt(), obs_out(), obs_rbind(), obs_read_csvy(), obs_roundtime(), obs_trunc(), rtorf-deprecated, sr()

Examples

```
## Not run:
# Do not run
df <- data.frame(a = rnorm(n = 10),
                 time = Sys.time() + 1:10)

f <- paste0(tempfile(), ".csvy")
notes <- c("notes",
           "more notes")
obs_write_csvy(dt = df, notes = notes, out = f)
readLines(f)
data.table::fread(f, h = TRUE)

## End(Not run)
```

sr *Extacts n last characters*

Description

file extension

Usage

```
sr(x, n)
```

Arguments

x	a character vector.
n	integer.

Value

last n characters

See Also

Other helpers: fex(), obs_footname(), obs_format(), obs_freq(), obs_list.dt(), obs_out(), obs_rbind(), obs_read_csvy(), obs_roundtime(), obs_trunc(), obs_write_csvy(), rtorf-deprecated

Examples

```
## Not run:
# do not run

## End(Not run)
```

Index