Package: lpjmlkit (via r-universe)

August 25, 2024

Type Package

Title Toolkit for Basic LPJmL Handling

Version 1.7.1

Description A collection of basic functions to facilitate the work with the Dynamic Global Vegetation Model (DGVM) Lund-Potsdam-Jena managed Land (LPJmL) hosted at the Potsdam Institute for Climate Impact Research (PIK). It provides functions for performing LPJmL simulations, as well as reading, processing and writing model-related data such as inputs and outputs or configuration files.

License AGPL-3

LazyData true

RoxygenNote 7.3.2

Roxygen list(markdown = TRUE, r6 = TRUE)

Encoding UTF-8

Depends R (>= 3.5.0)

URL https://github.com/PIK-LPJmL/lpjmlkit, https://doi.org/10.5281/zenodo.7773134

BugReports https://github.com/PIK-LPJmL/lpjmlkit/issues

Imports magrittr, dplyr, processx, tibble, jsonlite, doParallel, foreach, utils, methods, abind, rlang, withr, grDevices, cli, stringi

Suggests rmarkdown, knitr, testthat (>= 3.0.0), terra, raster, reshape2, maps, sf

Config/testthat/edition 3

VignetteBuilder knitr

Date 2024-07-26

Repository https://pik-piam.r-universe.dev

RemoteUrl https://github.com/PIK-LPJmL/lpjmlkit

RemoteRef HEAD

RemoteSha 051c06930a658347c72765cc03502cde17ec0bc6

Contents

lpjmlkit-package *lpjmlkit: Toolkit for Basic LPJmL Handling*

Description

A collection of basic functions to facilitate the work with the Dynamic Global Vegetation Model (DGVM) Lund-Potsdam-Jena managed Land (LPJmL) hosted at the Potsdam Institute for Climate Impact Research (PIK). It provides functions for performing LPJmL simulations, as well as reading, processing and writing model-related data such as inputs and outputs or configuration files.

Author(s)

Maintainer: Jannes Breier <jannesbr@pik-potsdam.de> (ORCID)

Authors:

- Sebastian Ostberg <ostberg@pik-potsdam.de> (ORCID)
- Stephen Björn Wirth <wirth@pik-potsdam.de> (ORCID)
- Sara Minoli <minoli@pik-potsdam.de> (ORCID)
- Fabian Stenzel <stenzel@pik-potsdam.de> (ORCID)
- David Hötten <davidho@pik-potsdam.de>
- Christoph Müller <cmueller@pik-potsdam.de> (ORCID)

See Also

Useful links:

- https://github.com/PIK-LPJmL/lpjmlkit
- doi:10.5281/zenodo.7773134
- Report bugs at https://github.com/PIK-LPJmL/lpjmlkit/issues

add_grid *Add grid to an LPJmLData object*

Description

Function to add a grid to an LPJmLData object. The function acts as a read_io() wrapper for the grid file and adds it as an LPJmLData object itself to the $grid attribute of the main object.

Usage

```
add_grid(x, ...)
```

Arguments

x	LPJmLData object.
...	Arguments passed to read_io(). Without any arguments, add_grid() will search for a file name starting with "grid" in the same directory that x was loaded from. This supports grid files in "meta" and "clm" format. If the grid file is in "raw" format or should be loaded from a different directory, supply all necessary read_io() parameters.

Details

Important:

- If "file_type" == "raw" prescribe variable = "grid" to ensure data are recognized as a grid.
- Do not use read_io() argument subset here. add_grid will use the subset of the parent LPJmLData object x.

Value

A copy of x (LPJmLData object) with added $grid attribute.

Examples

```
## Not run:

# Read in vegetation carbon data with meta file
vegc <- read_io(filename = "./vegc.bin.json")

# Add grid as attribute (via grid file in output directory)
vegc_with_grid <- add_grid(vegc)

## End(Not run)
```

---

asub *Subset a named array*

Description

Subset an array with the supplied dimnames and - if defined - replace values.

Usage

```
asub(x, ..., drop = TRUE)
```

Arguments

x	An array with named dimensions.
...	One or several vectors of indices or character strings to be used to subset x. Argument names refer to the dimension name to be subset, while argument values specify the selected elements along the respective dimension. Examples: cell = c(27411:27416), band = -c(14:16, 19:32), band = c("rainfed rice","rainfed maize").
drop	Logical. If TRUE (default), dimensions with a length of 1 are dropped from the result. Otherwise, they are kept.

Value

array (or vector if drop = TRUE and only one dimension is left) of the selected subset of x.

Examples

```
my_array <- array(1,
                  dim = c(cell = 67, month = 12, band = 3),
                  dimnames = list(cell = 0:66,
                                  month = 1:12,
                                  band = c("band1", "band2", "band3")))
my_subset <- asub(my_array,
                  band = c("band1", "band3"))
dimnames(my_subset)[3]
# $ band
#   [1] "band1"
#   [2] "band3"
```

---

as_array *Coerce an LPJmLData object to an array*

Description

Function to coerce (convert) an LPJmLData object into a pure array. Pure - because LPJmLData stores the data already as an array which can be accessed via $data. as_array provides additional functionality to subset or aggregate the array.

Usage

```
as_array(x, subset = NULL, aggregate = NULL, ...)
```

Arguments

x	LPJmLData object.
subset	List of array dimension(s) as name/key and corresponding subset vector as value, e.g. list(cell = c(27411:27415). More information at subset.LPJmLData().

aggregate	List of array dimension(s) as name/key and corresponding aggregation function as value, e.g. `list(band = sum)`.
...	Arguments passed to the aggregate function(s), e.g. `na.rm = TRUE`.

Value

an array with dimensions of object `$data` with applied `subset` and `aggregate` functionality as well as `dim` and `dimnames` from the `LPJmLData` object.

Examples

```
## Not run:

vegc <- read_io(filename = "./vegc.bin.json")

# Returns array attribute of LPJmLData object directly
vegc$data
#        time
# cell      1901-12-31   1902-12-31   1903-12-31   1904-12-31   1905-12-31
#   0     1.362730e+04 1.363163e+04 1.364153e+04 1.365467e+04 1.366689e+04
#   1     1.201350e+02 1.158988e+02 1.101675e+02 1.214204e+02 1.062658e+02
#   2     1.334261e+02 1.210387e+02 1.218128e+02 1.183210e+02 1.159934e+02
#   3     9.744530e+01 9.586801e+01 8.365642e+01 8.193731e+01 7.757602e+01
#   4     7.592700e+01 7.821202e+01 6.798551e+01 6.632317e+01 5.691082e+01
#   5     1.106748e+01 1.137272e+01 1.196524e+01 1.131316e+01 9.924266e+0

# Returns two-dimensional array with timeseries for the mean across cells
# 27410:27415
as_array(vegc,
         subset = list(cell = 27410:27415),
         aggregate = list(cell = mean))

#             band
# time                1
#   1901-12-31 1995.959
#   1902-12-31 1979.585
#   1903-12-31 1978.054
#   1904-12-31 1935.623
#   1905-12-31 1938.805


## End(Not run)
```

---

as_header *Coerce LPJmLMetaData to an LPJmL header object*

Description

Function to coerce (convert) an `LPJmLMetaData` object into an LPJmL header object. More information at `create_header()`.

Usage

```
as_header(x, silent = FALSE)
```

Arguments

x	An LPJmLMetaData object
silent	Logical. Whether to suppress notifications from header conversion/initialization.

Value

An LPJmL header object. More information at `create_header()`.

Examples

```
## Not run:

vegc_meta <- read_meta(filename = "./vegc.bin.json")

# Returns a list object with the structure of an LPJmL header
as_header(vegc_meta)
# $name
# [1] "LPJDUMMY"
#
# $header
#      version        order    firstyear        nyear    firstcell
#          4.0          4.0       1901.0        200.0          0.0
#        ncell       nbands cellsize_lon       scalar cellsize_lat
#      67420.0          1.0          0.5          1.0          0.5
#     datatype        nstep     timestep
#          3.0          1.0          1.0
#
# $endian
# [1] "little"


## End(Not run)
```

---

as_list *Coerce LPJmLMetaData to a list*

Description

Function to coerce (convert) an `LPJmLMetaData` object into a list.

Usage

```
as_list(x)
```

Arguments

x	An LPJmLMetaData object

Value

A list

Examples

```
## Not run:

vegc_meta <- read_meta(filename = "./vegc.bin.json")

# Returns one dimensional array with timeseries for cells `27410:27415`
as_list(vegc_meta)
# $sim_name
# [1] "lu_cf"
#
# $source
# [1] "LPJmL C Version 5.3.001"
#
# $variable
# [1] "vegc"
#
# $descr
# [1] "vegetation carbon"
#
# $unit
# [1] "gC/m2"
#
# $nbands
# [1] 1
#
# ...


## End(Not run)
```

---

as_raster *Coerce an LPJmLData object to a raster object*

Description

Function to coerce (convert) an LPJmLData object into a raster or brick object that allows for any GIS-based raster operations. Read more about the raster package at https://rspatial.github.io/raster/reference/raster-package.html. The successor package of raster is called terra: https://rspatial.org/.

Usage

```
as_raster(x, subset = NULL, aggregate = NULL, ...)
```

Arguments

x	LPJmLData object
subset	List of array dimension(s) as name/key and corresponding subset vector as value, e.g.`list(cell = c(27411:27415))`. More information at `subset.LPJmLData()`.
aggregate	List of array dimension(s) as name/key and corresponding aggregation function as value, e.g. `list(band = sum)`.
...	Arguments passed to the aggregate function(s), e.g. `na.rm = TRUE`.

Details

The `$grid` attribute is required for spatial transformation. When using `file_type = "meta"`, grid data are usually read automatically via `add_grid()` if the grid file is present in the same directory. Otherwise, `add_grid()` has to be called explicitly with the path to a matching grid file. Supports either multiple bands or multiple time steps. Use `subset` or `aggregate` to reduce data with multiple bands and time steps.

Value

A raster or brick object with spatial extent and coordinates based on internal `$grid` attribute and containing a lon/lat representation of `x$data`. If multiple bands or time steps exist, a brick is created. Further meta information such as the lon/lat resolution are extracted from `$meta`.

Examples

```
## Not run:

vegc <- read_io(filename = "./vegc.bin.json")

# Returns a RasterBrick for all data
as_raster(vegc)
# class      : RasterBrick
# dimensions : 280, 720, 201600, 200  (nrow, ncol, ncell, nlayers)
# resolution : 0.5, 0.5  (x, y)
# extent     : -180, 180, -56, 84  (xmin, xmax, ymin, ymax)
# crs        : +proj=longlat +datum=WGS84 +no_defs
# source     : memory
# names      : X1901.12.31, X1902.12.31, X1903.12.31, X1904.12.31, ...
# min values :           0,           0,           0,           0, ...
# max values :    28680.72,    28662.49,    28640.29,    28634.03, ...


## End(Not run)
```

as_terra *Coerce an LPJmLData object to a terra object*

Description

Function to coerce (convert) an LPJmLData object into a rast object that allows GIS-based raster operations. Read more about the terra package at https://rspatial.org/.

Usage

```
as_terra(x, subset = NULL, aggregate = NULL, ...)
```

Arguments

x	LPJmLData object.
subset	List of array dimension(s) as name/key and corresponding subset vector as value, e.g. list(cell = c(27411:27415). More information at subset.LPJmLData().
aggregate	List of array dimension(s) as name/key and corresponding aggregation function as value, e.g. list(band = sum).
...	Arguments passed to the aggregate function(s), e.g. na.rm = TRUE.

Details

The $grid attribute is required for spatial transformation. When using file_type = "meta", grid data are usually read automatically via add_grid() if the grid file is present in the same directory. Otherwise, add_grid() has to be called explicitly with the path to a matching grid file. Supports either multiple bands or multiple time steps. Use subset or aggregate to reduce data with multiple bands and time steps.

Value

A rast object with spatial extent and coordinates based on internal $grid attribute and containing a lon/lat representation of x$data. Further meta information such as the lon/lat resolution is extracted from $meta.

Examples

```
## Not run:

vegc <- read_io(filename = "./vegc.bin.json")

# Returns a SpatRaster for all data
as_terra(vegc)
# ...

## End(Not run)
```

`as_tibble.LPJmLData` *Coerce an LPJmLData object to a tibble*

Description

Function to coerce (convert) an `LPJmLData` object into a tibble (modern data.frame). Read more about tibbles at `https://r4ds.had.co.nz/tibbles.html`. Please make sure to call `lpjmlkit::as_tibble()` explicitly when also using the tidyverse packages tibble or dplyr.

Usage

```
## S3 method for class 'LPJmLData'
as_tibble(x, subset = NULL, aggregate = NULL, value_name = "value", ...)
```

Arguments

`x`	LPJmLData object
`subset`	List of array dimension(s) as name/key and corresponding subset vector as value, e.g. `list(cell = c(27411:27415))`. More information at `subset.LPJmLData()`.
`aggregate`	List of array dimension(s) as name/key and corresponding aggregation function as value, e.g. `list(band = sum)`.
`value_name`	Name of value column in returned `tibble`. Defaults to `"value"`.
`...`	Arguments passed to the aggregate function(s), e.g. `na.rm = TRUE`.

Value

a tibble with columns corresponding to dimension naming of the `LPJmLData$data` array and values in one value column.

Examples

```
## Not run:

vegc <- read_io(filename = "./vegc.bin.json")

# Returns two-dimensional tibble representation of vegc$data.
as_tibble(vegc)
#   cell  time       band    value
#   <fct> <fct>      <fct>   <dbl>
# 1 0     1901-12-31 1     13627.
# 2 1     1901-12-31 1       120.
# 3 2     1901-12-31 1       133.
# 4 3     1901-12-31 1        97.4
# 5 4     1901-12-31 1        75.9
# 6 5     1901-12-31 1        11.1


## End(Not run)
```

`calc_cellarea` *Calculate the cell area of LPJmL cells*

Description

Calculate the cell area of LPJmL cells based on an `LPJmLData` object or latitude coordinates and grid resolution. Uses a spherical representation of the Earth.

Usage

```
calc_cellarea(
  x,
  cellsize_lon = 0.5,
  cellsize_lat = cellsize_lon,
  earth_radius = 6371000.785,
  return_unit = "m2"
)
```

Arguments

`x`	`LPJmLData` object with `$grid` attribute, an LPJmLData object of variable `"grid"` (`"LPJGRID"`) or a vector of cell-center latitude coordinates in degrees.
`cellsize_lon`	Grid resolution in longitude direction in degrees (default: `0.5`). If `x` is an LPJmLData object the resolution will be taken from the meta data included in `x` if available.
`cellsize_lat`	Grid resolution in latitude direction in degrees (default: same as `cellsize_lon`). If `x` is an LPJmLData object the resolution will be taken from the meta data included in `x` if available.
`earth_radius`	Radius of the sphere (in m) used to calculate the cell areas.
`return_unit`	Character string describing the area unit of the returned cell areas. Defaults to `"m2"`, further options: `"ha"` or `"km2"`.

Value

A vector or array matching the space dimension(s) of `x` if `x` is an LPJmLData object. A vector of the same length as `x` if `x` is a vector of latitude coordinates. Cell areas are returned in the unit `return_unit`.

Examples

```
grid <- matrix(
  data = c(-179.75, 89.75, -0.25, 0.25, 0.25, -0.25, 179.75, -89.75),
  ncol = 2,
  byrow = TRUE,
  dimnames = list(NULL, c("lon", "lat"))
)
gridarea <- calc_cellarea(grid[,"lat"])
```

check_config *Check the validity of LPJmL config JSON files*

Description

Check if created LPJmL config JSON files (write_config()) are valid and are ready to be used for simulations using lpjcheck for multiple files.

Usage

```
check_config(
  x,
  model_path = ".",
  sim_path = NULL,
  return_output = FALSE,
  raise_error = FALSE,
  output_path = NULL
)
```

Arguments

x	job_details object returned by write_config() or character vector providing the config file names (hint: returns x as a list entry).
model_path	Character string providing the path to LPJmL (equal to LPJROOT environment variable). Defaults to ".".
sim_path	Character string defining path where all simulation data are written, including output, restart and configuration files. If NULL, model_path is used. See also write_config
return_output	Parameter affecting the output. If FALSE print stdout/stderr message. If TRUE, return the result of the check. Defaults to FALSE.
raise_error	Logical. Whether to raise an error if sub-process has non-zero exit status. Defaults to FALSE.
output_path	Argument is deprecated as of version 1.0; use sim_path instead.

Value

NULL.

Examples

```
## Not run:
library(tibble)
library(lpjmlkit)

model_path <- "./LPJmL_internal"
sim_path <-"./my_runs"
```

```
# Basic usage
my_params <- tibble(
  sim_name = c("scen1", "scen2"),
  random_seed = c(12, 404),
  `pftpar[[1]]$name` = c("first_tree", NA),
  `param$k_temp` = c(NA, 0.03),
  new_phenology = c(TRUE, FALSE)
)

config_details <- write_config(
  x = my_params,
  model_path = model_path,
  sim_path = sim_path
)

check_config(x = config_details,
  model_path = model_path,
  sim_path = sim_path,
  return_output = FALSE
)

## End(Not run)
```

---

create_header *Create a new LPJmL input/output file header*

Description

Create a header from scratch in the format required by `write_header()`.

Usage

```
create_header(
  name = "LPJGRID",
  version = 3,
  order = 1,
  firstyear = 1901,
  nyear = 1,
  firstcell = 0,
  ncell,
  nbands = 2,
  cellsize_lon = 0.5,
  scalar = 1,
  cellsize_lat = cellsize_lon,
  datatype = 3,
  nstep = 1,
  timestep = 1,
```

```
  endian = .Platform$endian,
  verbose = TRUE
)
```

Arguments

name	Header name attribute (default: '"LPJGRID").
version	CLM version to use (default: 3).
order	Order of data items. See details below or LPJmL code for supported values. The order may be provided either as an integer value or as a character string (default: 1).
firstyear	Start year of data in file (default: 1901).
nyear	Number of years of data included in file (default: 1).
firstcell	Index of first data item (default: 0).
ncell	Number of data items per band.
nbands	Number of bands per year of data (default: 2).
cellsize_lon	Longitude cellsize in degrees (default: 0.5).
scalar	Conversion factor applied to data when it is read by LPJmL or by read_io() (default: 1.0).
cellsize_lat	Latitude cellsize in degrees (default: same as cellsize_lon).
datatype	LPJmL data type in file. See details below or LPJmL code for valid data type codes (default: 3).
nstep	Number of time steps per year. Added in header version 4 to separate time bands from content bands (default: 1).
timestep	If larger than 1, outputs are averaged over timestep years and only written once every timestep years (default: 1).
endian	Endianness to use for file (either "big" or "little", by default uses platform-specific endianness .Platform$endian).
verbose	If TRUE (the default), function provides some feedback on datatype and when using default values for missing parameters. If FALSE, only errors are reported.

Details

File headers in input files are used by LPJmL to determine the structure of the file and how to read it. They can also be used to describe the structure of output files.

Header names usually start with "LPJ" followed by a word or abbreviation describing the type of input/output data. See LPJmL code for valid header names.

The version number determines the amount of header information included in the file. All versions save the header name and header attributes 'version', 'order', 'firstyear', 'nyear', 'firstcell', 'ncell', and 'nbands'. Header versions 2, 3 and 4 add header attributes 'cellsize_lon' and 'scalar'. Header versions 3 and 4 add header attributes 'cellsize_lat' and 'datatype'. Header version 4 adds attributes 'nstep' and 'timestep'.

Valid values for `order` are `1` / `"cellyear"`, `2` / `"yearcell"`, `3` / `"cellindex"`, and `4` / `"cellseq"`. The default for LPJmL input files is `1`. The default for LPJmL output files is `4`, except for grid output files which also use `1`.

By default, input files contain data for all cells, indicated by setting the `firstcell` index to `0`. If `firstcell` > `0`, LPJmL assumes the first `firstcell` cells to be missing in the data.

Valid codes for the `datatype` attribute and the corresponding LPJmL data types are: `0` / `"byte"` (LPJ_BYTE), `1` / `"short"` (LPJ_SHORT), `2` / `"int"` (LPJ_INT), `3` / `"float"` (LPJ_FLOAT), `4` / `"double"` (LPJ_DOUBLE).

The default parameters of the function are valid for grid input files using LPJ_FLOAT data type.

Value

The function returns a list with 3 components:

- name: The header name, e.g. "LPJGRID".
- header: Vector of header values ('version', 'order', 'firstyear', 'nyear', 'firstcell', 'ncell', 'nbands', 'cellsize_lon', 'scalar', 'cellsize_lat', 'datatype', 'nstep', 'timestep').
- endian: Endian used to write binary data, either "little" or "big".

See Also

- `read_header()` for reading headers from LPJmL input/output files.
- `write_header()` for writing headers to files.

Examples

```
header <- create_header(
  name = "LPJGRID",
  version = 3,
  order = 1,
  firstyear = 1901,
  nyear = 1,
  firstcell = 0,
  ncell = 67420,
  nbands = 2,
  cellsize_lon = 0.5,
  scalar = 1.0,
  cellsize_lat = 0.5,
  datatype = 3,
  nstep = 1,
  timestep = 1,
  endian = .Platform$endian,
  verbose = TRUE
)
```

`detect_io_type` *Detect the file type of an LPJmL input/output file*

Description

This utility function tries to detect automatically if a provided file is of `"clm"`, `"meta"`, or `"raw"` file type. NetCDFs and simple text formats such as ".txt" or ".csv" are also detected.

Usage

```
detect_io_type(filename)
```

Arguments

`filename`	Character string naming the file to check.

Value

Character vector of length 1 giving the file type:

- "cdf" for a NetCDF file (classic or NetCDF4/HDF5 format).
- "clm" for a binary LPJmL input/output file with header.
- "meta" for a JSON meta file describing a binary LPJmL input/output file.
- "raw" for a binary LPJmL input/output file without header. This is also the default if no other file type can be recognized.
- "text" for any type of text-only file, e.g. ".txt" or ".csv"

Examples

```
## Not run:
detect_io_type(filename = "filename.clm")
# [1] "clm"

## End(Not run)
```

dim.LPJmLData *Dimensions of an LPJmLData data array*

Description

Function to get the dimensions of the data array of an LPJmLData object.

Usage

```
## S3 method for class 'LPJmLData'
dim(x)
```

Arguments

x	LPJmLData object

Value

For the default method, either NULL or a numeric vector, which is coerced to integer (by truncation).

dimnames.LPJmLData *Dimnames of an LPJmLData data array*

Description

Function to get the dimnames (list) of the data array of an LPJmLData object.

Usage

```
## S3 method for class 'LPJmLData'
dimnames(x)
```

Arguments

x	LPJmLData object

Value

A list of the same length as dim(x). Components are character vectors with positive length of the respective dimension of x.

find_varfile *Search for a variable file in a directory*

Description

Function to search for a file containing a specific variable in a specific directory.

Usage

```
find_varfile(searchdir, variable = "grid", strict = FALSE)
```

Arguments

searchdir	Directory where to look for the variable file.
variable	Single character string containing the variable to search for
strict	Boolean. If set to TRUE, file must be named "variable.", where "" is one or two file extensions with 3 or 4 characters, e.g. "grid.bin.json" if variable = "grid". If set to FALSE, the function will first try to match the strict pattern. If unsuccessful, any filename that starts with "variable" will be matched.

Details

This function looks for file names in searchdir that match the pattern parameter in its list.files() call. Files of type "meta" are preferred. Files of type "clm" are also accepted. The function returns an error if no suitable file or multiple files are found.

Value

Character string with the file name of a matched file, including the full path.

get_cellindex *Get Cell Index*

Description

This function returns the cell index from a grid file based on the provided extent or coordinates. If neither extent nor coordinates are provided, the full grid will be returned. If both extent and coordinates are provided, the function will stop and ask for only one of them. The extent should be a vector of length 4 in the form c(lonmin, lonmax, latmin, latmax). If the extent is not in the correct form, the function will swap the values to correct it.

Usage

```
get_cellindex(grid_filename, extent = NULL, coordinates = NULL)
```

Arguments

grid_filename	A string representing the grid file name.
extent	A numeric vector (lonmin, lonmax, latmin, latmax) containing the longitude and latitude boundaries between which values included in the subset.
coordinates	A list of two named (lon, lat) numeric vectors representing the coordinates.

Details

The function reads a grid file specified by grid_filename and creates a data frame with columns for longitude, latitude, and cell number. The cell number is a sequence from 1 to the number of rows in the data frame.

If an extent is provided, the function filters the cells to include only those within the specified longitude and latitude range. The extent should be a numeric vector of length 4 in the form c(lonmin, lonmax, latmin, latmax).

If a list of coordinates is provided, the function filters the cells to include only those that match the specified coordinates. The coordinates should be a list of two character vectors representing the longitude and latitude values as for subset().

If both extent and coordinates are provided, the function will stop and ask for only one of them. If neither extent nor coordinates are provided, the function will return the cell numbers for all cells in the grid.

The function also includes checks for input types and values, and gives specific error messages for different error conditions. For example, it checks if the grid_filename exists, if the extent vector has the correct length, and if the coordinates list contains two vectors of equal length.

Value

The cell index from the grid file based on the provided extent or coordinates.

Examples

```
## Not run:
get_cellindex(
  grid_filename = "my_grid.bin.json",
  extent = c(-123.25, -122.75, 49.25, 49.75) # (lonmin, lonmax, latmin, latmax)
)
get_cellindex(
  grid_filename = "my_grid.bin.json",
  coordinates = list(lon = c(-123.25, -122.75), lat = c(49.25, 49.75))
)

## End(Not run)
```

get_datatype *Data type of an LPJmL input/output file*

Description

Provides information on the data type used in an LPJmL input/output file based on the 'datatype' attribute included in the file header.

Usage

```
get_datatype(header, fail = TRUE)
```

Arguments

header	Header list object as returned by read_header() or create_header(). Alternatively, can be a single integer just giving the data type code or a single character string giving one of the LPJmL type names c("byte", "short", "int", "float", "double").
fail	Determines whether the function should fail if the datatype is invalid (default: TRUE).

Value

On success, the function returns a list object with three components:

- type: R data type; can be used with what parameter of readBin().
- size: size of data type; can be used with size parameter of readBin().
- signed: whether or not the data type is signed; can be used with signed parameter of readBin().

If fail = FALSE, the function returns NULL if an invalid datatype is provided.

See Also

- read_header() for reading headers from LPJmL input/output files.
- create_header() for creating headers from scratch.
- get_headersize() for determining the size of file headers.

Examples

```
## Not run:
# Read file header
header <- read_header("filename.clm")
# Open file for reading
fp <- file("filename.clm", "rb")
# Skip over file header
seek(fp, get_headersize(header))
# Read in file data
```

```
file_data <- readBin(
  fp,
  what = get_datatype(header)$type,
  size = get_datatype(header)$size,
  signed = get_datatype(header)$signed,
  n = header$header["ncell"] * header$header["nbands"] *
      header$header["nyear"] * header$header["nstep"],
  endian = header[["endian"]]
)
# Close file
close(fp)

## End(Not run)
```

`get_headersize` *Determine the size of an LPJmL input/output file header*

Description

Returns the size in bytes of an LPJmL file header based on a header list object read by `read_header()` or generated by `create_header()`.

Usage

```
get_headersize(header)
```

Arguments

`header`	Header list object as returned by `read_header()` or `create_header()`.

Value

Integer value giving the size of the header in bytes. This can be used when seeking in the file or to calculate the expected total file size in combination with the number of included data values and the data type.

See Also

- `read_header()` for reading a header from an LPJmL input/output file.
- `create_header()` for creating a header from scratch.

Examples

```
## Not run:
header <- read_header("filename.clm")
size <- get_headersize(header)
# Open file for reading
fp <- file("filename.clm", "rb")
```

```
# Skip over file header
seek(fp, size)
# Add code to read data from file

## End(Not run)
```

`get_header_item` *Retrieve information from an LPJmL input/output file header*

Description

Convenience function to extract information from a header object as returned by `read_header()` or `create_header()`. Returns one item per call.

Usage

```
get_header_item(header, item)
```

Arguments

`header`	LPJmL file header as returned by `read_header()` or `create_header()`.
`item`	Header information item to retrieve. One of `c("name", "version", "order", "firstyear", "nyear", "firstcell", "ncell", "nbands", "cellsize_lon", "scalar", "cellsize_lat", "datatype", "nstep", "timestep", "endian")`.

Value

Requested header item. Character string in case of "name" and "endian", otherwise numeric value.

See Also

- `create_header()` for creating headers from scratch and for a more detailed description of the LPJmL header format.
- `read_header()` for reading headers from LPJmL input/output files.

Examples

```
## Not run:
# Read file header
header <- read_header("filename.clm")
nyear <- get_header_item(header = header, item = "nyear")

## End(Not run)
```

length.LPJmLData *Length of an LPJmLData data array*

Description

Function to get the length of the data array of an LPJmLData object.

Usage

```
## S3 method for class 'LPJmLData'
length(x)
```

Arguments

x	LPJmLData object

Value

A non-negative integer or numeric (which will be rounded down).

LPJmLData *LPJmL data class*

Description

A data container for LPJmL input and output. Container - because an LPJmLData object is an environment in which the data array as well as the meta data are stored after `read_io()`. The data array can be accessed via `$data`, the meta data via `$meta`. The enclosing environment is locked and cannot be altered by any other than the available modify methods to ensure its integrity and validity. Use base stats methods like `print()`, `summary.LPJmLData()` or `plot.LPJmLData()` to get insights and export methods like `as_tibble()` or `as_raster()` to export it into common working formats.

Active bindings

`meta` `LPJmLMetaData` object to store corresponding meta data.

`data` array containing the underlying data.

`grid` Optional `LPJmLData` object containing the underlying grid.

Methods

Public methods:

- LPJmLData$add_grid()
- LPJmLData$subset()
- LPJmLData$transform()
- LPJmLData$as_array()
- LPJmLData$as_tibble()
- LPJmLData$as_raster()
- LPJmLData$as_terra()
- LPJmLData$plot()
- LPJmLData$length()
- LPJmLData$dim()
- LPJmLData$dimnames()
- LPJmLData$summary()
- LPJmLData$print()
- LPJmLData$.__set_data__()
- LPJmLData$.__set_grid__()
- LPJmLData$new()
- LPJmLData$clone()

Method `add_grid()`**:** Method to add a grid to an `LPJmLData` object. See also `add_grid`

Usage:

```
LPJmLData$add_grid(...)
```

Arguments:

`...` See `add_grid()`.

Method `subset()`**:** Method to use dimension names of `LPJmLData$data` array directly to subset each dimension to match the supplied vectors.

Usage:

```
LPJmLData$subset(...)
```

Arguments:

`...` See `subset.LPJmLData()`

Method `transform()`**:** Method to transform inner `LPJmLData$data` array into another space or time format.

Usage:

```
LPJmLData$transform(...)
```

Arguments:

`...` See `transform()`.

Method `as_array()`**:** Method to coerce (convert) an `LPJmLData` object into an array.

Usage:

```
LPJmLData$as_array(...)
```

Arguments:

`...` See `as_array()`.

Method `as_tibble()`**:** Method to coerce (convert) an `LPJmLData` object into a tibble (modern data.frame).

Usage:

```
LPJmLData$as_tibble(...)
```

Arguments:

`...` See `as_tibble()`.

Method `as_raster()`**:** Method to coerce (convert) an `LPJmLData` object into a raster or brick object that can be used for any GIS-based raster operations.

Usage:

```
LPJmLData$as_raster(...)
```

Arguments:

`...` See `as_raster()`.

Method `as_terra()`**:** Method to coerce (convert) an `LPJmLData` object into a rast object that can be used for any GIS-based raster operations.

Usage:

```
LPJmLData$as_terra(...)
```

Arguments:

`...` See `as_terra()`.

Method `plot()`**:** Method to plot a time-series or raster map of an `LPJmLData` object.

Usage:

```
LPJmLData$plot(...)
```

Arguments:

`...` See `plot.LPJmLData()`.

Method `length()`**:** Method to get the length of the data array of an `LPJmLData` object.
See also length.

Usage:

```
LPJmLData$length()
```

Method `dim()`**:** Method to get the dimensions of the data array of an `LPJmLData` object.
See also dim.

Usage:

```
LPJmLData$dim()
```

Method `dimnames()`**:** Method to get the dimnames (list) of the data array of an `LPJmLData` object.

Usage:

```
LPJmLData$dimnames(...)
```

Arguments:

`...` See `dimnames.LPJmLData()`.

Method `summary()`**:** Method to get the summary of the data array of an `LPJmLData` object.

Usage:

```
LPJmLData$summary(...)
```

Arguments:

`...` See [`summary.LPJmLData()`].

Method `print()`**:** Method to print the `LPJmLData` object.
See also print.

Usage:

```
LPJmLData$print()
```

Method `.__set_data__()`**:** !Internal method only to be used for package development!

Usage:

```
LPJmLData$.__set_data__(data)
```

Arguments:

`data` Data array.

Method `.__set_grid__()`**:** !Internal method only to be used for package development!

Usage:

```
LPJmLData$.__set_grid__(grid)
```

Arguments:

`grid` An `LPJmLData` object holding grid coordinates.

Method `new()`**:** !Internal method only to be used for package development!

Usage:

```
LPJmLData$new(data, meta_data = NULL)
```

Arguments:

`data` `array` with LPJmL data.
`meta_data` An `LPJmLMetaData` object.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LPJmLData$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

LPJmLGridData *LPJmL grid data class*

Description

A dedicated data class for an LPJmL input or output grid. LPJmLGridData serves the spatial reference for any LPJmLData objects and matches its spatial dimensions ("cell" or "lon", "lat") when attached as an grid attribute to it.\ LPJmLGridData holds the information which longitude and latitude correspond to each cell center assuming WGS84 as the coordinate reference system or the corresponding cell index when the data comes with longitude and latitude dimension. As in LPJmLData the data array can be accessed via `$data`, the meta data via `$meta`.

Super class

`lpjmlkit::LPJmLData` -> `LPJmLGridData`

Methods

Public methods:

- `LPJmLGridData$add_grid()`
- `LPJmLGridData$plot()`
- `LPJmLGridData$new()`
- `LPJmLGridData$print()`
- `LPJmLGridData$clone()`

Method `add_grid()`**:** ! Not allowed to add a grid to an `LPJmLGridData` object.

Usage:

```
LPJmLGridData$add_grid(...)
```

Arguments:

`...` See `add_grid()`.

Method `plot()`**:** ! No plot function available for `LPJmLGridData` object. Use `as_raster()` or `as_terra()` (and `plot()`) to visualize the grid.

Usage:

```
LPJmLGridData$plot(...)
```

Arguments:

`...` See `plot()`.

Method `new()`**:** !Internal method only to be used for package development!

Usage:

```
LPJmLGridData$new(lpjml_data)
```

Arguments:

`lpjml_data` LPJmLData object with variable `"grid"`, `"cellid"` or `"LPJGRID"`

Method `print()`**:** Method to print the `LPJmLGridData`.
See also print

Usage:

```
LPJmLGridData$print()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LPJmLGridData$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`LPJmLMetaData` *LPJmL meta data class*

Description

A meta data container for LPJmL input and output meta data. Container - because an `LPJmLMetaData` object is an environment in which the meta data are stored after `read_meta()` (or `read_io()`). Each attribute can be accessed via `$<attribute>`. To get an overview over available attributes, `print` the object or export it as a list `as_list()`. The enclosing environment is locked and cannot be altered.

Active bindings

`sim_name` Simulation name (works as identifier in LPJmL Runner).

`source` LPJmL version (character string).

`history` Character string of the call used to run LPJmL. This normally includes the path to the LPJmL executable and the path to the configuration file for the simulation.

`variable` Name of the input/output variable, e.g. `"npp"` or `"runoff"`.

`descr` Description of the input/output variable.

`unit` Unit of the input/output variable.

`nbands` Number (numeric) of bands (categoric dimension). Please note that `nbands` follows the convention in LPJmL, which uses the plural form for bands as opposed to `nyear` or `ncell`.

`band_names` Name of the bands (categoric dimension). Not included if `nbands = 1`.

`nyear` Number (numeric) of data years in the parent `LPJmLData` object.

`firstyear` First calendar year (numeric) in the parent `LPJmLData` object.

`lastyear` Last calendar year (numeric) in the parent `LPJmLData` object.

`nstep` Number (numeric) of intra-annual time steps. `1` for annual, `12` for monthly, and `365` for daily data.

`timestep` Number (numeric) of years between time steps. `timestep = 5` means that output is written every 5 years.

`ncell` Number (numeric) of cells in the parent `LPJmLData` object.

`firstcell` First cell (numeric) in the parent `LPJmLData` object.

`cellsize_lon` Longitude cellsize in degrees (numeric).

`cellsize_lat` Latitude cellsize in degrees (numeric).

`datatype` File data type (character string), e.g. `"float"`. Note that data are converted into R-internal data type by `read_io()`.

`scalar` Conversion factor (numeric) applied when reading raw data from file. The parent `LPJmLData` object contains the values after the application of the conversion factor.

`order` Order of the data items in the file, either `"cellyear"`, `"yearcell"`, `"cellindex"`, or `"cellseq"`. The structure of the data array in the parent `LPJmLData` object may differ from the original order in the file depending on the `dim_order` parameter used in `read_io()`.

`offset` Offset (numeric) at the start of the binary file before the actual data start.

`bigendian` (Logical) Endianness refers to the order in which bytes are stored in a multi-byte value, with big-endian storing the most significant byte at the lowest address and little-endian storing the least significant byte at the lowest address.

`format` Binary format (character string) of the file containing the actual data. Either `"raw"`, `"clm"` (raw with header), or `"cdf"` for NetCDF format.

`filename` Name of the file containing the actual data.

`subset` Logical. Whether parent `LPJmLData` object is subsetted.

`map` Character vector describing how to map the bands in an input file to the bands used inside LPJmL. May be used by `read_io()` to construct a `band_names` attribute.

`version` Version of data file.

`._data_dir_` *Internal* character string containing the directory from which the file was loaded.

`._subset_space_` *Internal* logical. Whether space dimensions are subsetted in the parent `LPJmLData` object.

`._fields_set_` *Internal* character vector of names of attributes set by the meta file.

`._time_format_` *Internal* character string describing the time dimension format, either `"time"` or `"year_month_day"`.

`._space_format_` *Internal* character string describing the space dimension format, either `"cell"` or `"lon_lat"`.

`._dimension_map_` *Internal* dictionary/list of space and time dimension formats with categories and namings.

Methods

Public methods:

- `LPJmLMetaData$as_list()`
- `LPJmLMetaData$as_header()`
- `LPJmLMetaData$print()`
- `LPJmLMetaData$.__init_grid__()`
- `LPJmLMetaData$.__update_subset__()`
- `LPJmLMetaData$.__transform_time_format__()`
- `LPJmLMetaData$.__transform_space_format__()`

- LPJmLMetaData$.__set_attribute__()
- LPJmLMetaData$new()
- LPJmLMetaData$clone()

Method `as_list()`**:** Method to coerce (convert) an `LPJmLMetaData` object into a list. See also `as_list()`.

Usage:

```
LPJmLMetaData$as_list()
```

Method `as_header()`**:** Method to coerce (convert) an `LPJmLMetaData` object into an LPJmL binary file header. More information about file headers at `create_header()`).

Usage:

```
LPJmLMetaData$as_header(...)
```

Arguments:

`...` See `as_header()`.

Method `print()`**:** Method to print an `LPJmLMetaData` object. See also print.

Usage:

```
LPJmLMetaData$print(all = TRUE, spaces = "")
```

Arguments:

`all` Logical. Should all attributes be printed or only the most relevant (`all = FALSE`)?

`spaces` *Internal parameter* Spaces to be printed at the start.

Method `.__init_grid__()`**:** !Internal method only to be used for package development!

Usage:

```
LPJmLMetaData$.__init_grid__()
```

Method `.__update_subset__()`**:** !Internal method only to be used for package development!

Usage:

```
LPJmLMetaData$.__update_subset__(
  subset,
  cell_dimnames = NULL,
  time_dimnames = NULL,
  year_dimnames = NULL
)
```

Arguments:

`subset` List of subset arguments, see also `subset.LPJmLData()`.

`cell_dimnames` Optional list of new cell_dimnames of subset data to update meta data. Required if spatial dimensions are subsetted.

`time_dimnames` Optional list of new time_dimnames of subset data to update meta data. Required if time dimension is subsetted.

`year_dimnames` Optional list of new year_dimnames of subset data to update meta data. Required if year dimension is subsetted.

Method `.__transform_time_format__()`: !Internal method only to be used for package development!

Usage:

```
LPJmLMetaData$.__transform_time_format__(time_format)
```

Arguments:

`time_format` Character. Choose between `"year_month_day"` and `"time"`.

Method `.__transform_space_format__()`: !Internal method only to be used for package development!

Usage:

```
LPJmLMetaData$.__transform_space_format__(space_format)
```

Arguments:

`space_format` Character. Choose between `"lon_lat"` and `"cell"`.

Method `.__set_attribute__()`: !Internal method only to be used for package development!

Usage:

```
LPJmLMetaData$.__set_attribute__(key, value)
```

Arguments:

`key` Name of the attribute, e.g. `"variable"`

`value` Value of the attribute, e.g. `"grid"`

Method `new()`: Create a new LPJmLMetaData object.

Usage:

```
LPJmLMetaData$new(x, additional_attributes = list(), data_dir = NULL)
```

Arguments:

`x` A list (not nested) with meta data.

`additional_attributes` A list of additional attributes to be set that are not included in file header or JSON meta file. These are `c"(band_names", "variable", "descr", "unit")`

`data_dir` Directory containing the file this LPJmLMetaData object refers to. Used to "lazy load" grid.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
LPJmLMetaData$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

make_lpjml *Compile LPJmL model*

Description

Compiles the LPJmL source code and creates an executable by executing "make all" on the operating system shell.

Usage

```
make_lpjml(
  model_path = ".",
  parallel_cores = NULL,
  make_clean = FALSE,
  raise_error = TRUE,
  debug = NULL
)
```

Arguments

model_path	Character string providing the path to LPJmL (equal to LPJROOT environment variable). Defaults sto ".".
parallel_cores	Numeric defining the number of available CPU cores for parallelization.
make_clean	Logical. If set to TRUE, calls "make clean" first to remove previous installation. Defaults to FALSE.
raise_error	Logical. Whether to raise an error if sub-process has non-zero exit status, hence if compilation fails. Defaults to TRUE.
debug	NULL or Logical. Whether to compile LPJmL with "-debug" flag. Defaults to NULL. If set to FALSE or TRUE, make_clean is set automatically and compilation configuration is reset with/without "-debug". Also the "configure.sh" file is rewritten.

Value

A list with process status, see run.

Examples

```
## Not run:
model_path <- "./LPJmL_internal"
make_lpjml(model_path = model_path)

## End(Not run)
```

`plot.LPJmLData` *Plot an LPJmLData object*

Description

Function to plot a time-series or raster map of an `LPJmLData` object.

Usage

```
## S3 method for class 'LPJmLData'
plot(x, subset = NULL, aggregate = NULL, raster_extent = NULL, ...)
```

Arguments

`x`	LPJmLData object
`subset`	List of array dimension(s) as name/key and corresponding subset vector as value, e.g. `list(cell = c(27411:27416))`. More information at `subset.LPJmLData()`.
`aggregate`	List of array dimension(s) as name/key and corresponding aggregation function as value, e.g. `list(band = sum)`.
`raster_extent`	Optional parameter to crop map display of spatial data. An extent or any object from which an Extent object can be extracted. Not relevant if `aggregate` includes spatial dimension.
`...`	Arguments passed to plot and plot

Details

Depending on the dimensions of the LPJmLData object's internal data array the plot will be a ...

- single map plot: more than 8 `"cell"`s or `"lat"` & `"lon"` dimensions available)
- multiple maps plot: length of one time (e.g.`"time"`, `"year"`, `"month"`) or `"band"` dimension > 1.
- time series plot: less than 9 `"cell"`s
- lat/lon plot: a subsetted/aggregated `"lat"` or `"lon"` dimension

The plot can only handle 2-3 dimensions. Use arguments `subset` and `aggregate` to modify `x$data` to the desired plot type. If more than three dimensions have length > 1,' plot will return an error and suggest to reduce the number of dimensions.

Note that the plot function aims to provide a quick overview of the data rather than create publication-ready graphs.

Value

No return value; called for side effects.

Examples

```
## Not run:

vegc <- read_io(filename = "./vegc.bin.json")

# Plot first 9 years starting from 1901 as a raster plot
plot(vegc)

# Plot raster with mean over the whole time series
plot(vegc,
     aggregate = list(time = mean))

# Plot only year 2010 as a raster
plot(vegc,
     subset = list(time = "2010"))

# Plot first 10 time steps as global mean time series. Note: Aggregation
# across cells is not area-weighted.
plot(vegc,
     subset = list(time = 1:10),
     aggregate = list(cell = mean))

# Plot time series for cells with LPJmL index 27410 - 27415 (C indices start
#    at 0 in contrast to R indices starting at 1).
plot(vegc,
     subset = list(cell = 27411:27416))


## End(Not run)
```

`read_config` *Read an LPJmL configuration file*

Description

Reads a configuration (config) file (compilable csjon/js file or json file) and turns it into a nested list object.

Usage

```
read_config(filename, from_restart = FALSE, macro = "")
```

Arguments

`filename`	Character string representing path (if different from current working directory) and filename.

from_restart	Logical defining whether config files should be read as from_restart (transient run) or without (spinup run). Defaults to FALSE (spinup run). Used only if file is not pre-compiled (no json).
macro	Optional character string to pass one or several macros to the pre-compiler, e.g. ("-DFROM_RESTART"). Used only if file is not pre-compiled (no json).

Value

A nested list object representing the LPJmL configuration read from filename.

Examples

```
## Not run:
 config <- read_config(filename = "config_spinup.json")

 config[["version"]]
 # [1] "5.3"

 config[["pftpar"]][[1]][["name"]]
 # [1] "tropical broadleaved evergreen tree"

 config[["input"]][["coord"]][["name"]]
 # [1] "input_VERSION2/grid.bin"

 # visualize configuration as tree view
 View(config)

## End(Not run)
```

---

read_grid *Read LPJmL input and output grid files*

Description

Generic function to read LPJmL input & output files in different formats. Depending on the format, arguments can be automatically detected or have to be passed as individual arguments.

Usage

```
read_grid(...)
```

Arguments

...	See read_io for further arguments.

Details

See read_io for more details.

Value

An LPJmLGridData object.

Examples

```
## Not run:
my_grid <- read_io("grid.bin.json")


## End(Not run)
```

`read_header` *Read header (any version) from LPJmL input/output file*

Description

Reads a header from an LPJmL clm file. CLM is the default format used for LPJmL input files and can also be used for output files.

Usage

```
read_header(filename, force_version = NULL, verbose = TRUE)
```

Arguments

`filename`	Filename to read header from.
`force_version`	Manually set clm version. The default value `NULL` means that the version is determined automatically from the header. Set only if the version number in the file header is incorrect.
`verbose`	If `TRUE` (the default), `read_header` provides some feedback when using default values for missing parameters. If `FALSE`, only errors are reported.

Value

The function returns a list with 3 components:

- name: Header name, e.g. "LPJGRID"; describes the type of data in the file.
- header: Vector of header values ('version', 'order', 'firstyear', 'nyear', 'firstcell', 'ncell', 'nbands', 'cellsize_lon', 'scalar', 'cellsize_lat', 'datatype', 'nstep', 'timestep') describing the file structure. If header version is <4, the header is partially filled with default values.
- endian: Endianness of file (`"little"` or `"big"`).

See Also

- `create_header()` for a more detailed description of the LPJmL header format.
- `write_header()` for writing headers to files.

Examples

```
## Not run:
header <- read_header("filename.clm")

## End(Not run)
```

`read_io` *Read LPJmL input and output files*

Description

Generic function to read LPJmL input & output files in different formats. Depending on the format, arguments can be automatically detected or have to be passed as individual arguments.

Usage

```
read_io(
  filename,
  subset = list(),
  band_names = NULL,
  dim_order = c("cell", "time", "band"),
  file_type = NULL,
  version = NULL,
  order = NULL,
  firstyear = NULL,
  nyear = NULL,
  firstcell = NULL,
  ncell = NULL,
  nbands = NULL,
  cellsize_lon = NULL,
  scalar = NULL,
  cellsize_lat = NULL,
  datatype = NULL,
  nstep = NULL,
  timestep = NULL,
  endian = NULL,
  variable = NULL,
  descr = NULL,
  unit = NULL,
  name = NULL,
  silent = FALSE
)
```

Arguments

filename	Mandatory character string giving the file name to read, including its path and extension.
subset	Optional list allowing to subset data read from the file along one or several of its dimensions. See details for more information.
band_names	Optional vector of character strings providing the band names or NULL. Normally determined automatically from the meta file in case of output files using file_type = "meta".
dim_order	Order of dimensions in returned LPJmLData object. Must be a character vector containing all of the following in any order: c("cell", "time", "band"). Users may select the order most useful to their further data processing.
file_type	Optional character string giving the file type. This is normally detected automatically but can be prescribed if automatic detection is incorrect. Valid options: • "raw", a binary file without header. • "clm", a binary file with header. • "meta", a meta information JSON file complementing a raw or clm file.
version	Integer indicating the clm file header version, currently supports one of c(1, 2, 3, 4).
order	Integer value or character string describing the order of data items in the file (default in input file: 1; in output file: 4). Valid values for LPJmL input/output files are "cellyear"/ 1, "yearcell" / 2, "cellindex"/ 3, and "cellseq" / 4, although only options 1 and 4 are supported by this function.
firstyear	Integer providing the first year of data in the file.
nyear	Integer providing the number of years of data included in the file. These are not consecutive in case of timestep > 1.
firstcell	Integer providing the cell index of the first data item. 0 by default.
ncell	Integer providing the number of data items per band.
nbands	Integer providing the number of bands per time step of data.
cellsize_lon	Numeric value providing the longitude cell size in degrees.
scalar	Numeric value providing a conversion factor that needs to be applied to raw data when reading it from file to derive final values.
cellsize_lat	Numeric value providing the latitude cell size in degrees.
datatype	Integer value or character string describing the LPJmL data type stored in the file. Supported options: "byte" / 0, "short" / 1, "int" / 2, "float" / 3, or "double" / 4.
nstep	Integer value defining the number of within-year time steps of the file. Valid values are 1 (yearly), 12 (monthly), 365 (daily). Defaults to 1 if not read from file ("clm" or "meta" file) or provided by the user.
timestep	Integer value providing the interval in years between years represented in the file data. Normally 1, but LPJmL also allows averaging annual outputs over several years. Defaults to 1 if not read from file ("clm" or "meta" file) or provided by user.

endian	Endianness to use for file (either "big" or "little"). By default uses endianness determined from file header or set in meta information or the platform-specific endianness .Platform$endian if not set.
variable	Optional character string providing the name of the variable contained in the file. Included in some JSON meta files. **Important:** If file_type == "raw", prescribe variable = "grid" to ensure that data are recognized as a grid.
descr	Optional character string providing a more detailed description of the variable contained in the file. Included in some JSON meta files.
unit	Optional character string providing the unit of the data in the file. Included in some JSON meta files.
name	Optional character string specifying the header name. This is usually read from clm headers for file_type = "clm" but can be specified for the other file_type options.
silent	If set to TRUE, suppresses most warnings or messages. Use only after testing that read_io() works as expected with the files it is being used on. Default: FALSE.

Details

The file_type determines which arguments are mandatory or optional. filename must always be provided. file_type is usually detected automatically. Supply only if detected file_type is incorrect.

In case of file_type = "meta", if any of the function arguments not listed as "mandatory" are provided and are already set in the JSON file, a warning is given, but they are still overwritten. Normally, you would only set meta attributes not set in the JSON file.

In case of file_type = "clm", function arguments not listed as "optional" are usually determined automatically from the file header included in the clm file. Users may still provide any of these arguments to overwrite values read from the file header, e.g. when they know that the values in the file header are wrong. Also, clm headers with versions < 4 do not contain all header attributes, with missing attributes filled with default values that may not be correct for all files.

In case of file_type = "raw", files do not contain any information about their structure. Users should provide all arguments not listed as "optional". Otherwise, default values valid for LPJmL standard outputs are used for arguments not supplied by the user. For example, the default firstyear is 1901, the default for nyear, nbands, nstep, and timestep is 1.

subset can be a list containing one or several named elements. Allowed names are "band", "cell", and "year".

- "year" can be used to return data for a subset of one or several years included in the file. Integer indices can be between 1 and nyear. If subsetting by actual calendar years (starting at firstyear) a character vector has to be supplied.
- "band" can be used to return data for a subset of one or several bands included in the file. These can be specified either as integer indices or as a character vector if bands are named.
- "cell" can be used to return data for a subset of cells. Note that integer indices start counting at 1, whereas character indices start counting at the value of firstcell (usually 0).

Value

An LPJmLData object.

Examples

```
## Not run:
# First case: meta file. Reads meta information from "my_file.json" and
# data from binary file linked in "my_file.json". Normally does not require
# any additional arguments.
my_data <- read_io("my_file.json")

# Suppose that file data has two bands named "wheat" and "rice". `band_names`
# are included in the JSON meta file. Select only the "wheat" band during
# reading and discard the "rice" band. Also, read only data for years
# 1910-1920.
my_data_wheat <- read_io(
  "my_file.json",
  subset = list(band = "wheat", year = as.character(seq(1910, 1920)))
)

# Read data from clm file. This includes a header describing the file
# structure.
my_data_clm <- read_io("my_file.clm")

# Suppose that "my_file.clm" has two bands containing data for "wheat" and
# "rice". Assign names to them manually since the header does not include a
# `band_names` attribute.
my_data_clm <- read_io("my_file.clm", band_names = c("wheat", "rice"))

# Once `band_names` are set, subsetting by name is possible also for
# file_type = "clm"
my_data_wheat <- read_io(
  "my_file.clm",
  band_names = c("wheat", "rice"),
  subset = list(band = "wheat", year = as.character(seq(1910, 1920)))
)

# Read data from raw binary file. All information about file structure needs
# to be supplied. Use default values except for nyear (1 by default), and
# nbands (also 1 by default).
my_data <- read_io("my_file.bin", nyear = 100, nbands = 2)

# Supply band_names to be able to subset by name
my_data_wheat <- read_io(
  "my_file.bin",
  band_names = c("wheat", "rice"), # length needs to correspond to `nbands`
  subset = list(band = "wheat", year = as.character(seq(1910, 1920))),
  nyear = 100,
  nbands = 2,
)

## End(Not run)
```

`read_meta` *Read an LPJmL meta file or binary file header*

Description

Reads a meta JSON file or the header of a binary LPJmL input or output file.

Usage

```
read_meta(filename, ...)
```

Arguments

filename	Character string representing path (if different from current working directory) and filename.
...	Additional arguments passed to read_header if header file is read.

Value

An LPJmLMetaData object.

Examples

```
## Not run:
 meta <- read_meta(filename = "mpft_npp.bin.json")

 meta$sim_name
 # [1] "LPJmL Run"

 meta$firstcell
 # [1] 27410

 meta$band_names[1]
 # [1] "tropical broadleaved evergreen tree"

## End(Not run)
```

---

run_lpjml *Run LPJmL model*

Description

Runs LPJmL using "config_*.json" files written by write_config(). write_config() returns a tibble that can be used as an input (see x). It contains the details to run single or multiple (dependent/subsequent) model runs.

Usage

```
run_lpjml(
  x,
  model_path = ".",
  sim_path = NULL,
  run_cmd = "srun --propagate",
  parallel_cores = 1,
  write_stdout = FALSE,
  raise_error = TRUE,
  output_path = NULL
)
```

Arguments

x	A tibble with at least one column named "sim_name". Each simulation gets a separate row. Optional run parameters are "order" and "dependency" which are used for subsequent simulations (see details). write_config() returns a tibble in the required format. OR provide a character string (vector) with the file name of one or multiple generated configuration file(s).
model_path	Character string providing the path to LPJmL (equal to LPJROOT environment variable). Defaults to "."
sim_path	Character string defining path where all simulation data are written, including output, restart and configuration files. If NULL, model_path is used. See also write_config
run_cmd	Character string defining the command used to execute lpjml (see details). Defaults to "srun –propagate" (compute ondes of old cluster at PIK). Change to "mpirun" for HPC2024 at PIK.
parallel_cores	Integer defining the number of available CPU cores/nodes for parallelization. Defaults to 1 (no parallelization). Please note that parallelization is only supported for SLURM jobs and not for interactive runs.
write_stdout	Logical. If TRUE, stdout as well as stderr files are written. If FALSE (default), these are printed instead. Within a SLURM job write_stdout is automatically set to TRUE.
raise_error	Logical. Whether to raise an error if sub-process has non-zero exit status. Defaults to TRUE.
output_path	Argument is deprecated as of version 1.0; use sim_path instead.

Details

x: A tibble for x that has been generated by write_config() and can look like the following examples can supplied:

sim_name
scen1_spinup
scen2_transient

To perform subsequent or rather dependent runs the optional run parameter "dependency" needs to be provided within the initial tibble supplied as param to write_config().

sim_name	order	dependency
scen1_spinup	1	NA
scen2_transient	2	scen1 _spinup

As a shortcut it is also possible to provide the config file "config_*.json" as a character string or multiple config files as a character string vector directly as the x argument to run_lpjml.
Also be aware that the order of the supplied config files is important (e.g. make sure the spin-up run is run before the transient one).

run_cmd: The run_cmd argument is used to define the command to execute LPJmL. This is needed because the LPJmL executable can not directly be used on all machines. Which command has to be used depends on the software installed. Further information on this can be found in the INSTALL file of LPJmL. To determine the correct command, check the lpj_submit.sh file in the bin directory of LPJmL. Using PIK infrastrucure the command is srun for the hpc2015 and mpirun for the hpc2024. To facilitate usage on the interactive (login) nodes, no command is needed for hpc2015. For the hpc2024 the command remains mpirun (in these cases run_lpjml adjusts run_cmd accordingly).

Value

See x, extended by columns "type", "job_id" and "status".

Examples

```
## Not run:
library(tibble)

model_path <- "./LPJmL_internal"
sim_path <-"./my_runs"

# Basic usage
my_params1 <- tibble(
  sim_name = c("scen1", "scen2"),
  startgrid = c(27410, 27410),
  river_routing = c(FALSE, FALSE),
  random_seed = c(42, 404),
  `pftpar[[1]]$name` = c("first_tree", NA),
  `param$k_temp` = c(NA, 0.03),
  new_phenology = c(TRUE, FALSE)
)

config_details1 <- write_config(my_params1, model_path, sim_path)

 run_details1 <- run_lpjml(
  x = config_details1,
  model_path = model_path,
  sim_path = sim_path
```

```
)

run_details1
#   sim_name      job_id   status
#   <chr>           <int>  <chr>
# 1 scen1              NA  run
# 2 scen2              NA  run


# With run parameters dependency and order being set (also less other
#   parameters than in previous example)
my_params2 <- tibble(
  sim_name = c("scen1", "scen2"),
  startgrid = c(27410, 27410),
  river_routing = c(FALSE, FALSE),
  random_seed = c(42, 404),
  dependency = c(NA, "scen1_spinup")
)

config_details2 <- write_config(my_params2, model_path, sim_path)

run_details2 <- run_lpjml(config_details2, model_path, sim_path)

run_details2
#   sim_name        order dependency   type       job_id   status
#   <chr>           <dbl> <chr>        <chr>      <chr>    <chr>
# 1 scen1_spinup        1 NA           simulation NA       run
# 2 scen1_transient     2 scen1_spinup simulation NA       run


# Same but by using the pipe operator
library(magrittr)

run_details2 <- tibble(
  sim_name = c("scen1_spinup", "scen1_transient"),
  random_seed = as.integer(c(1, 42)),
  dependency = c(NA, "scen1_spinup")
) %>%
  write_config(model_path, sim_path) %>%
  run_lpjml(model_path, sim_path)


# Shortcut approaches
run_details3 <- run_lpjml(
  x = "./config_scen1_transient.json",
  model_path = model_path,
  sim_path = sim_path
)

run_details4 <- run_lpjml(
  c("./config_scen1_spinup.json", "./config_scen1_transient.json"),
  model_path,
  sim_path
```

```
)

## End(Not run)
```

`set_header_item` *Set information in an LPJmL input (or output) file header*

Description

Convenience function to set information in a header object as returned by `read_header()` or `create_header()`. One or several

Usage

```
set_header_item(header, ...)
```

Arguments

`header`	An LPJmL file header as returned by `read_header()` or `create_header()`.
`...`	Named header items to set. Can be one or several of 'name', 'version', 'order', 'firstyear', 'nyear', 'firstcell', 'ncell', 'nbands', 'cellsize_lon', 'scalar', 'cellsize_lat', 'datatype', 'nstep', 'timestep', 'endian'. Parameter 'verbose' can be used to control verbosity, as in `create_header()`.

Value

Header `header` where header items supplied through the ellipsis have been changed.

See Also

- `create_header()` for creating headers from scratch and for a more detailed description of the LPJmL header format.
- `read_header()` for reading headers from files.

Examples

```
header <- create_header(
  name = "LPJGRID",
  version = 3,
  order = 1,
  firstyear = 1901,
  nyear = 1,
  firstcell = 0,
  ncell = 67420,
  nbands = 2,
  cellsize_lon = 0.5,
```

```
  scalar = 1.0,
  cellsize_lat = 0.5,
  datatype = 3,
  nstep = 1,
  timestep = 1,
  endian = .Platform$endian,
  verbose = TRUE
)

header
# $name
# [1] "LPJGRID"
#
# $header
#      version        order    firstyear        nyear    firstcell        ncell
#          3.0          1.0       1901.0          1.0          0.0      67420.0
#        nbands cellsize_lon       scalar cellsize_lat     datatype       nstep
#          2.0          0.5          1.0          0.5          3.0          1.0
#     timestep
#          1.0
#
# $endian
# [1] "little"

# Change number of cells to 1
set_header_item(header = header, ncell = 1)
# $name
# [1] "LPJGRID"
#
# $header
#      version        order    firstyear        nyear    firstcell        ncell
#          3.0          1.0       1901.0          1.0          0.0          1.0
#        nbands cellsize_lon       scalar cellsize_lat     datatype       nstep
#          2.0          0.5          1.0          0.5          3.0          1.0
#     timestep
#          1.0
#
# $endian
# [1] "little"
```

`submit_lpjml` *Submit LPJmL model simulation to SLURM*

Description

LPJmL simulations are submitted to SLURM using `"config*.json"` files written by `write_config()`. `write_config()` returns a tibble that can be used as an input (see `x`). It serves the details to submit single or multiple (dependent/subsequent) model simulations.

Usage

```
submit_lpjml(
  x,
  model_path,
  sim_path = NULL,
  group = "",
  sclass = "short",
  ntasks = 256,
  wtime = "",
  blocking = "",
  constraint = "",
  slurm_options = list(),
  no_submit = FALSE,
  output_path = NULL
)
```

Arguments

x	A tibble with at least one column named "sim_name". Each simulation gets a separate row. An optional run parameter "dependency" is used for subsequent simulations (see details). write_config() returns a tibble in the required format. OR provide a character string (vector) with the file name of one or multiple generated config file(s).
model_path	Character string providing the path to LPJmL (equal to LPJROOT environment variable).
sim_path	Character string defining path where all simulation data are written, including output, restart and configuration files. If NULL, model_path is used. See also write_config
group	Character string defining the user group for which the job is submitted.
sclass	Character string defining the job classification. Available options at PIK: c("short", "medium", "long", "priority", "standby", "io") More information at https://www.pik-potsdam.de/en. Defaults to "short".
ntasks	Integer defining the number of tasks/threads. More information at https://www.pik-potsdam.de/en and https://slurm.schedmd.com/sbatch.html. Defaults to 256.
wtime	Character string defining the time limit. Setting a lower time limit than the maximum runtime for sclass can reduce the wait time in the SLURM job queue. More information at https://www.pik-potsdam.de/en and https://slurm.schedmd.com/sbatch.html.
blocking	Integer defining the number of cores to be blocked. More information at https://www.pik-potsdam.de/en and https://slurm.schedmd.com/sbatch.html.
constraint	Character string defining constraints for node selection. Use constraint = "haswell" to request nodes of the type haswell with 16 cores per node, constraint = "broadwell" to request nodes of the type broadwell CPUs with 32 cores per node or constraint = "exclusive" to reserve all CPUs of assigned nodes

even if less are requested by ntasks. Using exclusive should prevent interference of other batch jobs with LPJmL. More information at https://www.pik-potsdam.de and https://slurm.schedmd.com/sbatch.html.

slurm_options — A named list of further arguments to be passed to sbatch. E.g. list(mail-user = "max.mustermann@pik-potsdam.de") More information at https://www.pik-potsdam.de and https://slurm.schedmd.com/sbatch.html

no_submit — Logical. Set to TRUE to test if x set correctly or FALSE to actually submit job to SLURM.

output_path — Argument is deprecated as of version 1.0; use sim_path instead.

Details

A tibble for x that has been generated by write_config() and can look like the following examples can supplied:

sim_name
scen1_spinup
scen2_transient

To perform subsequent or rather dependent simulations the optional run parameter "dependency" needs to be provided within the initial tibble supplied as param to write_config().

sim_name	dependency
scen1_spinup	NA
scen2_transient	scen1 _spinup

To use different SLURM settings for each run the optional SLURM options "sclass", "ntasks", "wtime", "blocking"orconstraintcan also be supplied to the initial \link[tibble]{tibble} supplied asparam tingor constraint) supplied to submit_lpjml.

sim_name	dependency	wtime
scen1_spinup	NA	"8:00:00"
scen2_transient	scen1 _spinup	"2:00:00"

As a shortcut it is also possible to provide the config file "config_*.json" as a character string or multiple config files as a character string vector directly as the x argument to submit_lpjml. With this approach, run parameters or SLURM options cannot be taken into account.

Value

See x, extended by columns "type", "job_id" and "status".

Examples

```
## Not run:
library(tibble)

model_path <- "./LPJmL_internal"
sim_path <-"./my_runs"


# Basic usage
my_params <- tibble(
 sim_name = c("scen1", "scen2"),
 random_seed = as.integer(c(42, 404)),
  `pftpar[[1]]$name` = c("first_tree", NA),
  `param$k_temp` = c(NA, 0.03),
 new_phenology = c(TRUE, FALSE)
)

config_details <- write_config(my_params, model_path, sim_path)

 run_details <- submit_lpjml(
  x = config_details,
  model_path = model_path,
  sim_path = sim_path
)

run_details
#   sim_name      job_id   status
#   <chr>           <int>  <chr>
# 1 scen1        21235215  submitted
# 2 scen2        21235216  submitted


# With run parameter dependency and SLURM option wtime being
#   set (also less other parameters than in previous example)
my_params <- tibble(
  sim_name = c("scen1", "scen2"),
  random_seed = as.integer(c(42, 404)),
  dependency = c(NA, "scen1_spinup"),
  wtime = c("8:00:00", "4:00:00"),
)

config_details2 <- write_config(my_params2, model_path, sim_path)

run_details2 <- submit_lpjml(config_details2, model_path, sim_path)

run_details2
#   sim_name        order dependency   wtime   type       job_id   status
#   <chr>           <dbl> <chr>        <chr>   <chr>      <chr>    <chr>
# 1 scen1_spinup        1 NA           8:00:00 simulation 22910240 submitted
# 2 scen1_transient     2 scen1_spinup 4:00:00 simulation 22910241 submitted
```

```
# Same but by using the pipe operator
library(magrittr)

run_details <- tibble(
  sim_name = c("scen1_spinup", "scen1_transient"),
  random_seed = as.integer(c(1, 42)),
  dependency = c(NA, "scen1_spinup"),
  wtime = c("8:00:00", "4:00:00"),
) %>%
  write_config(model_path, sim_path) %>%
  submit_lpjml(model_path, sim_path)


# Shortcut approach
run_details <- submit_lpjml(
  x = "./config_scen1_transient.json",
  model_path = model_path,
  sim_path = sim_path
)

run_details <- submit_lpjml(
  c("./config_scen1_spinup.json", "./config_scen1_transient.json"),
  model_path,
  sim_path
)


## End(Not run)
```

`subset.LPJmLData` *Subset an LPJmLData object*

Description

Function to extract a subset of the full data in an `LPJmLData` object by applying selections along one or several of its dimensions.

Usage

```
## S3 method for class 'LPJmLData'
subset(x, ...)
```

Arguments

`x`	An LPJmLData object
`...`	One or several key-value combinations where keys represent the dimension names and values represent the requested elements along these dimensions. Subsets may either specify integer indices, e.g. `cell = c(27411:27416)`, band =

`-c(14:16, 19:32)`, or character vectors if the dimension has a dimnames attribute, e.g. `band = c("rainfed rice", "rainfed maize")`.\ Coordinate pairs of individual cells can be selected by providing a list or tibble in the form of `coords = list(lon = ..., lat =...)`. Coordinate values need to be supplied as character vectors. The argument can also be called `coordinates`. When coordinates are supplied as character vectors to subset either along the `lon` or `lat` dimension or to subset by coordinate pair, the function matches the grid cells closest to the supplied coordinate value.

Value

An `LPJmLData` object with dimensions resulting from the selection in `subset`. Meta data are updated as well.

Examples

```
## Not run:

vegc <- read_io(filename = "./vegc.bin.json")

# Subset cells by index
subset(vegc, cell = seq(27410, 27415))
# [...]
# $data |>
#   dimnames() |>
#     .$cell  "27409" "27410" "27411" "27412" "27413" "27414"
#     .$time  "1901-12-31" "1902-12-31" "1903-12-31" "1904-12-31" ...
#     .$band  "1"
# [...]

# Subset time by character vector
subset(vegc, time = c("2001-12-31", "2002-12-31", "2003-12-31"))
# [...]
# $data |>
#   dimnames() |>
#     .$cell  "0" "1" "2" "3" ... "67419"
#     .$time  "2001-12-31" "2002-12-31" "2003-12-31"
#     .$band  "1"
# [...]

## End(Not run)
```

`summary.LPJmLData` *LPJmLData object summary*

Description

Function to get the summary of the data array of an LPJmLData object. See also summary.

Usage

```
## S3 method for class 'LPJmLData'
summary(object, ...)
```

Arguments

`object`	LPJmLData object
`...`	Further arguments: • `dimension` for which a summary is printed for every element (in style of matrix summary). Default is `dimension = "band"`. Choose from available dimensions like `"time"` or `"cell"`. • `subset` list of array dimension(s) as name/key and corresponding subset vector as value, e.g. `list(cell = c(27411:27415)`. More information at `subset.LPJmLData()`. • `cutoff` (logical) If `TRUE` summary for dimension elements > 16 are cut off. • Additional arguments to be passed on to summary.

Value

Summary for object of class matrix (see summary) for selected dimension(s) and if defined subset.

`transform` *Transform an LPJmLData object*

Description

Function to transform an `LPJmLData` data object into another space or another time format. Combinations of space and time formats are also possible.

Usage

```
transform(x, to)
```

Arguments

`x`	An LPJmLData object.
`to`	A character vector defining space and/or time format into which the corresponding data dimensions should be transformed. Choose from space formats `c("cell", "lon_lat")` and time formats `c("time","year_month_day")`.

Value

An `LPJmLData` object in the selected format.

Examples

```
## Not run:

runoff <- read_io(filename = "runoff.bin.json",
                  subset = list(year = as.character(1991:2000)))

# Transform into space format "lon_lat". This assumes a "grid.bin.json" file
# is present in the same directory as "runoff.bin.json".
transform(runoff, to = "lon_lat")
# [...]
# $data |>
#   dimnames() |>
#     .$lat  "-55.75" "-55.25" "-54.75" "-54.25" ... "83.75"
#     .$lon  "-179.75" "-179.25" "-178.75" "-178.25" ... "179.75"
#     .$time  "1991-01-31" "1991-02-28" "1991-03-31" "1991-04-30" ...
#     .$band  "1"
# [...]

# Transform time format from a single time dimension into separate dimensions
# for years, months, and days. Dimensions for time steps not present in the
# data are omitted, i.e. no "day" dimension for monthly data.
transform(runoff, to = "year_month_day")
# [...]
# $data |>
#   dimnames() |>
#     .$lat  "-55.75" "-55.25" "-54.75" "-54.25" ... "83.75"
#     .$lon  "-179.75" "-179.25" "-178.75" "-178.25" ... "179.75"
#     .$month  "1" "2" "3" "4" ... "12"
#     .$year  "1991" "1992" "1993" "1994" ... "2000"
#     .$band  "1"
# [...]

## End(Not run)
```

`write_config` *Write LPJmL config files (JSON)*

Description

Requires a tibble (modern data.frame class) in a specific format (see details & examples) to write the model configuration file `"config_*.json"`. Each row in the tibble corresponds to a model run. The generated `"config_*.json"` is based on a cjson file (e.g. `"lpjml_config.cjson"`).

Usage

```
write_config(
  x,
  model_path,
```

```
  sim_path = NULL,
  output_list = c(),
  output_list_timestep = "annual",
  output_format = NULL,
  cjson_filename = "lpjml_config.cjson",
  parallel_cores = 4,
  debug = FALSE,
  params = NULL,
  output_path = NULL,
  js_filename = NULL
)
```

Arguments

x	A tibble in a defined format (see details).
model_path	Character string providing the path to LPJmL (equal to LPJROOT environment variable).
sim_path	Character string defining path where all simulation data are written. Also an output, a restart and a configuration folder are created in sim_path to store respective data. If NULL, model_path is used.
output_list	Character vector containing the "id" of outputvars. If defined, only these defined outputs will be written. Otherwise, all outputs set in cjson_filename will be written. Defaults to NULL.
output_list_timestep	Single character string or character vector defining what temporal resolution the defined outputs from output_list should have. Either provide a single character string for all outputs or a vector with the length of output_list defining each timestep individually. Choose between "annual", "monthly" or "daily".
output_format	Character string defining the format of the output. Defaults to NULL (use default from cjson file). Options: "raw", "cdf" (NetCDF) or "clm" (file with header).
cjson_filename	Character string providing the name of the main LPJmL configuration file to be parsed. Defaults to "lpjml_config.cjson".
parallel_cores	Integer defining the number of available CPU cores for parallelization. Defaults to 4.
debug	logical If TRUE, the inner parallelization is switched off to enable tracebacks and all types of error messages. Defaults to FALSE.
params	Argument is deprecated as of version 1.0; use x instead.
output_path	Argument is deprecated as of version 1.0; use sim_path instead.
js_filename	Argument is deprecated as of version 1.3; use cjson_filename instead.

Details

Supply a tibble for x, in which each row represents a configuration (config) for an LPJmL simulation.
Here a config refers to a precompiled "lpjml_config.cjson" file (or file name provided as cjson_filename

argument) which already contains all the information from the mandatory cjson files. The precompilation is done internally by `write_config()`.
`write_config()` uses the column names of `param` as keys for the config json using the same syntax as lists, e.g. `"k_temp"` from `"param.js"` can be accessed with `"param$k_temp"` or `"param[["k_temp"]]"` as the column name. (The former point-style syntax - `"param.k_temp"` - is still valid but deprecated)
For each run and thus each row, this value has to be specified in the tibble. If the original value should instead be used, insert `NA`.
Each run can be identified via the `"sim_name"`, which is mandatory to specify.

```
my_params1 <- tibble(
  sim_name = c("scenario1", "scenario2"),
  random_seed = c(42, 404),
  `pftpar[[1]]$name` = c("first_tree", NA),
  `param$k_temp` = c(NA, 0.03),
  new_phenology = c(TRUE, FALSE)
)

my_params1
# A tibble: 2 x 5
#   sim_name random_seed `pftpar[[1]]$name` `param$k_temp` new_phenology
#   <chr>          <dbl> <chr>                 <dbl> <lgl>
# 1 scenario1         42 first_tree             NA    TRUE
# 2 scenario2        404 NA                      0.03 FALSE
```

Simulation sequences:
To set up spin-up and transient runs, where transient runs are dependent on the spin-up(s), a parameter `"dependency"` has to be defined as a column in the tibble that links simulations with each other using the `"sim_name"`.
Do not manually set "-DFROM_RESTART" when using `"dependency"`. The same applies for LPJmL config settings "restart", "write_restart", "write_restart_filename", "restart_filename", which are set automatically by this function. This way multiple runs can be performed in succession and build a conceivably endless chain or tree.

```
# With dependent runs.
my_params3 <- tibble(
 sim_name = c("scen1_spinup", "scen1_transient"),
 random_seed = c(42, 404),
 dependency = c(NA, "scen1_spinup")
)
my_params3
# A tibble: 2 x 4
#   sim_name        random_seed order dependency
#   <chr>                 <int> <lgl> <chr>
# 1 scen1_spinup             42 FALSE NA
# 2 scen1_transient         404 TRUE  scen1_spinup
```

SLURM options:
Another feature is to define SLURM options for each simulation (row) separately. For example, users may want to set a lower wall clock limit (`wtime`) for the transient run than the spin-up run

to get a higher priority in the SLURM queue. This can be achieved by supplying this option as a parameter to `param`.
6 options are available, namely `sclass`, `ntasks`, `wtime`, `blocking`, `constraint` and `slurm_options`.
Use as arguments for [submit_lpjml()].\cr If specified in param, they overwrite the corresponding function a

```
my_params4 <- tibble(
 sim_name = c("scen1_spinup", "scen1_transient"),
 random_seed = c(42, 404),
 dependency = c(NA, "scen1_spinup"),
 wtime = c("8:00:00", "2:00:00")
)

my_params4
# A tibble: 2 x 5
#   sim_name        random_seed order dependency   wtime
#   <chr>                 <int> <lgl> <chr>        <chr>
# 1 scen1_spinup             42 FALSE NA           8:00:00
# 2 scen1_transient         404 TRUE  scen1_spinup 2:00:00
```

Use of macros:

To set a macro (e.g. "MY_MACRO" or "CHECKPOINT") provide it as a column of the tibble as you would do with a flag in the shell: `"-DMY_MACRO" "-DCHECKPOINT"`.
Wrap macros in backticks or tibble will raise an error, as starting an object definition with `"-"` is not allowed in *R*.

```
my_params2 <- tibble(
  sim_name = c("scen1_spinup", "scen1_transient"),
  random_seed = c(42, 404),
  `-DMY_MACRO` = c(TRUE, FALSE),
)

my_params2
# A tibble: 2 x 3
#   sim_name        random_seed `-DMY_MACRO`
#   <chr>                 <int> <lgl>
# 1 scen1_spinup             42 TRUE
# 2 scen1_transient         404 FALSE
```

In short:

- `write_config()` creates subdirectories within the `sim_path` directory
  - `"./configurations"` to store the config files.
  - `"./output"` to store the output within subdirectories for each `sim_name`.
  - `"./restart"` to store the restart files within subdirectories for each `sim_name`.
- The list syntax (e.g. `pftpar[[1]]$name`) allows to create column names and thus keys for accessing values in the config json.
- The column `"sim_name"` is mandatory (used as an identifier).
- The run parameter `"dependency"` is optional but enables interdependent consecutive runs using `submit_lpjml()`.

- SLURM options in `param` allow to use different values per run.
- If `NA` is specified as cell value the original value is used.
- *R* booleans/logical constants `TRUE` and `FALSE` are to be used for boolean parameters in the config json.
- Value types need to be set correctly, e.g. no strings where numeric values are expected.

Value

tibble with at least one column named `"sim_name"`. Run parameters `"order"` and `"dependency"` are included if defined in `x`. tibble in this format is required for `submit_lpjml()`.

Examples

```
## Not run:
library(tibble)

model_path <- "./LPJmL_internal"
sim_path <-"./my_runs"


# Basic usage
my_params <- tibble(
  sim_name = c("scen1", "scen2"),
  random_seed = c(12, 404),
  `pftpar[[1]]$name` = c("first_tree", NA),
  `param$k_temp` = c(NA, 0.03),
  new_phenology = c(TRUE, FALSE)
)

config_details <- write_config(
  x = my_params,
  model_path = model_path,
  sim_path = sim_path
)

config_details
# A tibble: 2 x 1
#   sim_name
#   <chr>
# 1 scen1
# 2 scen2

# Usage with dependency
my_params <- tibble(
 sim_name = c("scen1_spinup", "scen1_transient"),
 random_seed = c(42, 404),
 dependency = c(NA, "scen1_spinup")
)

config_details <- write_config(
  x = my_params,
  model_path = model_path,
```

```
  sim_path = sim_path
)

config_details
# A tibble: 2 x 3
#   sim_name        order dependency
#   <chr>           <dbl> <chr>
# 1 scen1_spinup        1 NA
# 2 scen1_transient     2 scen1_spinup


my_params <- tibble(
 sim_name = c("scen1_spinup", "scen1_transient"),
 random_seed = c(42, 404),
 dependency = c(NA, "scen1_spinup"),
 wtime = c("8:00:00", "2:00:00")
)

config_details <- write_config(
  x = my_params,
  model_path = model_path,
  sim_path = sim_path
)

config_details
# A tibble: 2 x 4
#   sim_name        order dependency   wtime
#   <chr>           <dbl> <chr>        <chr>
# 1 scen1_spinup        1 NA           8:00:00
# 2 scen1_transient     2 scen1_spinup 2:00:00


## End(Not run)
```

`write_header` *Write LPJmL header object to an LPJmL input (or output) file*

Description

Write an LPJmL clm header to a file. The header has to be a list following the structure returned by `read_header()` or `create_header()`. The function will fail if the output file exists already unless `overwrite` is set to `TRUE`.

Usage

```
write_header(filename, header, overwrite = FALSE)
```

Arguments

filename	Filename to write header into.
header	The header to be written.
overwrite	Whether to overwrite an existing output file (default FALSE).

Value

Returns filename invisibly.

See Also

- create_header() for creating headers from scratch and for a more detailed description of the LPJmL header format.
- read_header() for reading headers from files.

Examples

```
## Not run:
header <- read_header(filename = "old_filename.clm")
write_header(
  filename = "new_filename.clm",
  header = header,
  overwrite = FALSE
)

## End(Not run)
```

Index