Package ‘stars’

June 8, 2021

Title Spatiotemporal Arrays, Raster and Vector Data Cubes
Version 0.5-3
Description Reading, manipulating, writing and plotting
spatiotemporal arrays (raster and vector data cubes) in 'R', using 'GDAL'
bindings provided by 'sf', and 'NetCDF' bindings by 'ncmeta' and 'RNetCDF'.
License Apache License
URL https://r-spatial.github.io/stars/,
https://github.com/r-spatial/stars/
BugReports https://github.com/r-spatial/stars/issues/
Additional_repositories http://gis-bigdata.uni-muenster.de/pebesma/
LazyData true
Depends R (>= 3.3.0), abind, sf (>= 0.9-7)
Imports methods, parallel, classInt (>= 0.4-1), lwgeom, rlang, units
Suggests PCICt, RNetCDF (>= 1.8-2), clue, covr, cubelyr, digest, dplyr
(>= 0.7-0), exactextractr, future.apply, ggforce, ggplot2,
ggthemes, gstat, httr, jsonlite, knitr, maps, mapdata,
cncdfgeom, ncmeta (>= 0.0.3), pbapply, plm, randomForest,
raster, rgdal, rmarkdown, s2 (>= 1.0.0), sp, spacetime,
spatstat (>= 2.0-1), spatstat.geom, starsdata, terra, testthat,
tidyR, viridis, xts, zoo
VignetteBuilder knitr
Encoding UTF-8
RoxygenNote 7.1.1
NeedsCompilation no
Author  Edzer Pebesma [aut, cre] (<https://orcid.org/0000-0001-8049-7069>),
        Michael Sumner [ctb] (<https://orcid.org/0000-0002-2471-7511>),
        Etienne Racine [ctb],
        Adriano Fantini [ctb],
        David Blodgett [ctb]

Maintainer  Edzer Pebesma <edzer.pebesma@uni-muenster.de>

Repository  CRAN

Date/Publication  2021-06-08 14:50:02 UTC

R topics documented:

aggregate.stars .................................................. 3
as ................................................................. 5
bcsd_obs .......................................................... 5
c.stars ............................................................ 6
contour.stars ...................................................... 7
cut_stars .......................................................... 8
dplyr ............................................................... 9
geom_stars ........................................................ 10
L7_ETMs ............................................................ 11
make_intervals .................................................... 11
merge ............................................................... 12
ops_stars .......................................................... 12
plot ............................................................... 14
predict.stars ...................................................... 17
read_ncdf ........................................................ 17
read_stars ........................................................ 19
redimension ......................................................... 22
stars_sentinel2 ..................................................... 22
stars_subset ....................................................... 23
st_apply ........................................................... 24
st_as_sf ........................................................... 26
st_as_stars ....................................................... 28
st_contour ......................................................... 31
st_coordinates ..................................................... 31
st_crop ............................................................ 32
st_dimensions ....................................................... 34
st_dim_to_attr ..................................................... 36
st_extract ........................................................ 37
st_intersects.stars ................................................ 38
st_join.stars ...................................................... 39
st_mosaic .......................................................... 40
st_rasterize ....................................................... 41
st_raster_type ..................................................... 42
st_rgb ............................................................. 43
st_set_bbox ......................................................... 44
st_sfc2xy .......................................................... 44
aggregate.stars

spatially or temporally aggregate stars object

Description

spatially or temporally aggregate stars object, returning a data cube with lower spatial or temporal resolution

Usage

## S3 method for class 'stars'
aggregate(
x, by, FUN, ..., drop = FALSE, join = st_intersects,
as_points = any(st_dimension(by) == 2, na.rm = TRUE),
rightmost.closed = FALSE, left.open = FALSE, exact = FALSE
)

Arguments

x object of class stars with information to be aggregated

by object of class sf or sfc for spatial aggregation, for temporal aggregation a vector with time values (Date, POSIXct, or PCICT) that is interpreted as a sequence of left-closed, right-open time intervals or a string like "months", "5 days" or the like (see cut.POSIXt); if by is an object of class stars, it is converted to sfc by st_as_sfc(by, as_points = FALSE) thus ignoring its time component.

FUN aggregation function, such as mean

... arguments passed on to FUN, such as na.rm=TRUE

drop logical; ignored

join function; function used to find matches of x to by

as_points see st_as_sf: shall raster pixels be taken as points, or small square polygons?
rightmost.closed
 see findInterval

left.open logical; used for time intervals, see findInterval and cut.POSIXt

exact logical; if TRUE, use coverage_fraction to compute exact overlap fractions of polygons with raster cells

Examples

# aggregate time dimension in format Date
tif = system.file("tif/L7_ETMs.tif", package = "stars")
t1 = as.Date("2018-07-31")
x = read_stars(c(tif, tif, tif, tif), along = list(time = c(t1, t1+1, t1+2, t1+3)))[,1:30,1:30]
st_get_dimension_values(x, "time")
x_agg_time = aggregate(x, by = t1 + c(0, 2, 4), FUN = max)

# aggregate time dimension in format Date - interval
by_t = "2 days"
x_agg_time2 = aggregate(x, by = by_t, FUN = max)
st_get_dimension_values(x_agg_time2, "time")
x_agg_time - x_agg_time2

# aggregate time dimension in format POSIXct
x = st_set_dimensions(x, 4, values = as.POSIXct(c("2018-07-31",
  "2018-08-01", "2018-08-02", "2018-08-03")), names = "time")
by_t = as.POSIXct(c("2018-07-31", "2018-08-02"))
x_agg_posix = aggregate(x, by = by_t, FUN = max)
st_get_dimension_values(x_agg_posix, "time")
x_agg_time - x_agg_posix
aggregate(x, "2 days", mean)

# Spatial aggregation, see https://github.com/r-spatial/stars/issues/299
prec_file = system.file("nc/test_stageiv_xyt.nc", package = "stars")
prec = read_ncdf(prec_file, curvilinear = c("lon", "lat"))
prec_slice = dplyr::slice(prec, index = 17, along = "time")
nc = sf::read_sf(system.file("gpkg/nc.gpkg", package = "sf"), "nc.gpkg")
nc = st_transform(nc, st_crs(prec_slice))
agg = aggregate(prec_slice, st_geometry(nc), mean)
plot(agg)

# example of using a function for "by": aggregate by month-of-year
d = c(10, 10, 150)
a = array(rnorm(prod(d)), d) # pure noise
times = Sys.Date() + seq(1, 2000, length.out = d[3])
m = as.numeric(format(times, "%m"))
signal = rep(sin(m / 12 * pi), each = prod(d[1:2])) # yearly period
s = (st_as_stars(a) + signal) %>%
  st_set_dimensions(3, values = times)
f = function(x, format = "%B") {
  months = format(as.Date(paste0("01-", 1:12, "-1970")), format)
  factor(format(x, format), levels = months)
as

agg = aggregate(s, f, mean)
plot(agg)

---

**as**

*Coerce stars object into a Raster raster or brick*

**Description**

Coerce stars object into a Raster raster or brick

**Arguments**

from

object to coerce

**Details**

If the stars object has more than three dimensions, all dimensions higher than the third will be collapsed into the third dimensions. If the stars object has only an x/y raster but multiple attributes, these are merged first, then put in a raster brick.

**Value**

RasterLayer or RasterBrick

---

**bcasd_obs**

*Monthly Gridded Meteorological Observations*

**Description**

These are the monthly observational data used for BCSD downscaling. See: [http://gdo-dcp.ucnl.org/downscaled_cmip_project](http://gdo-dcp.ucnl.org/downscaled_cmip_project) for more information.

"Atmospheric Temperature, Air Temperature Atmosphere, Precipitation, Rain, Maximum Daily Temperature, Minimum Daily Temperature";

**Usage**

bcasd_obs

**Format**

An object of class stars_proxy (inherits from stars) of dimension 81 x 33 x 12.
c.stars  
combine multiple stars objects, or combine multiple attributes in a single stars object into a single array

Description
combine multiple stars objects, or combine multiple attributes in a single stars object into a single array

Usage
## S3 method for class 'stars'
c(
  ..., 
  along = NA_integer_, 
  try_hard = FALSE, 
  nms = names(list(...)), 
  tolerance = sqrt(.Machine$double.eps) 
)

## S3 method for class 'stars_proxy'
c(
  ..., 
  along = NA_integer_, 
  along_crs = FALSE, 
  try_hard = FALSE, 
  nms = names(list(...)),
  tolerance = sqrt(.Machine$double.eps) 
)

Arguments

...  object(s) of class star: in case of multiple arguments, these are combined into a single stars object, in case of a single argument, its attributes are combined into a single attribute. In case of multiple objects, all objects should have the same dimensionality.

along  integer; see read_stars

try_hard  logical; if TRUE and some arrays have different dimensions,

nms  character; vector with array names

tolerance  numeric; values used in all.equal to compare dimension values combine those that dimensions matching to the first array

along_crs  logical; if TRUE, combine arrays along a CRS dimension
contour.stars

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
(new = c(x, x))
c(new) # collapses two arrays into one with an additional dimension
(c(x, x, along = 3))

contour.stars  plot contours of a stars object

Description
plot contours of a stars object

Usage
## S3 method for class 'stars'
contour(x, ...)

Arguments
x object of class stars
...
other parameters passed on to contour

Details
this uses the R internal contour algorithm, which (by default) plots contours; st_contour uses the
GDAL contour algorithm that returns contours as simple features.

Examples

d = st_dimensions(x = 1:ncol(volcano), y = 1:nrow(volcano))
r = st_as_stars(t(volcano))
r = st_set_dimensions(r, 1, offset = 0, delta = 1)
r = st_set_dimensions(r, 2, offset = 0, delta = -1)
plot(r, reset = FALSE)
contour(r, add = TRUE)
cut_methods for stars objects

Description

cut methods for stars objects

Usage

## S3 method for class 'array'
cut(x, breaks, ...)

## S3 method for class 'matrix'
cut(x, breaks, ...)

## S3 method for class 'stars'
cut(x, breaks, ...)

Arguments

x see cut
breaks see cut
... see cut

Details

R’s factor only works for vectors, not for arrays or matrices. This is a work-around (or hack?) to keep the factor levels generated by cut and use them in plots.

Value

an array or matrix with a levels attribute; see details

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
cut(x, c(0, 50, 100, 255))
cut(x[,,,1], c(0, 50, 100, 255))
plot(cut(x[,,,1], c(0, 50, 100, 255)))
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x1 = read_stars(tif)
(x1_cut = cut(x1, breaks = c(0, 50, 100, Inf))) # shows factor in summary
plot(x1_cut[,,,c(3,6)]) # propagates through [ and plot
dplyr verbs for stars objects

Description

dplyr verbs for stars objects; package dplyr needs to be loaded before these methods can be used for stars objects.

Usage

filter.stars(.data, ...)
filter.stars_proxy(.data, ...)
mutate.stars(.data, ...)
mutate.stars_proxy(.data, ...)
transmute.stars(.data, ...)
transmute.stars_proxy(.data, ...)
select.stars(.data, ...)
select.stars_proxy(.data, ...)
pull.stars(.data, var = -1)
pull.stars_proxy(.data, ...)
as.tbl_cube.stars(x, ...)
slice.stars(.data, along, index, ..., drop = length(index) == 1)
slice.stars_proxy(.data, ...)
replace_na.stars(data, replace, ...)
replace_na.stars_proxy(data, ...)

Arguments

.data object of class stars
... see filter
var see pull
x object of class stars
along name or index of dimension to which the slice should be applied
index integer value(s) for this index
drop logical; drop dimensions that only have a single index?
data data set to work on
replace see replace_na: list with variable=value pairs, where value is the replacement value for NA's

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x1 = read_stars(tif)
library(dplyr)
x1 %>% slice("band", 2:3)
x1 %>% slice("x", 50:100)

geom_stars ggplot geom for stars objects

description

Usage

geom_stars(mapping = NULL, data = NULL, ..., downsample = 0, sf = FALSE)

theme_stars(...)

Arguments

mapping see geom_raster
data see geom_raster
... see geom_raster
downsample downsampling rate: e.g. 3 keeps rows and cols 1, 4, 7, 10 etc.; a value of 0 does not downsample; can be specified for each dimension, e.g. c(5, 5, 0) to downsample the first two dimensions but not the third.
sf logical; if TRUE rasters will be converted to polygons and plotted using geom_sf.

Details

geom_stars returns (a call to) either geom_raster, geom_tile, or geom_sf, depending on the raster or vector geometry; for the first to, an aes call is constructed with the raster dimension names and the first array as fill variable. Further calls to coord_equal and facet_wrap are needed to control aspect ratio and the layers to be plotted; see examples.
Examples

```r
system.file("tif/L7_ETMs.tif", package = "stars") %>% read_stars() -> x
library(ggplot2)
ggplot() + geom_stars(data = x) +
  coord_equal() +
  facet_wrap(~band) +
  theme_void() +
  scale_x_discrete(expand=c(0,0)) +
  scale_y_discrete(expand=c(0,0))
```

L7_ETMs

*Landsat-7 bands for a selected region around Olinda, BR*

Description

Probably containing the six 30 m bands:

- Band 1 Visible (0.45 - 0.52 µm) 30 m
- Band 2 Visible (0.52 - 0.60 µm) 30 m
- Band 3 Visible (0.63 - 0.69 µm) 30 m
- Band 4 Near-Infrared (0.77 - 0.90 µm) 30 m
- Band 5 Short-wave Infrared (1.55 - 1.75 µm) 30 m
- Band 7 Mid-Infrared (2.08 - 2.35 µm) 30 m

Usage

L7_ETMs

Format

An object of class `stars_proxy` (inherits from `stars`) of dimension 349 x 352 x 6.

make_intervals

create an intervals object

Description

create an intervals object, assuming left-closed and right-open intervals

Usage

make_intervals(start, end)

Arguments

- **start**: vector with start values, or 2-column matrix with start and end values in column 1 and 2, respectively
- **end**: vector with end values
merge

merge or split stars object

Description

merge attributes into a dimension, or split a dimension over attributes

Usage

## S3 method for class 'stars'
split(x, f = length(dim(x)), drop = TRUE, ...)

## S3 method for class 'stars'
merge(x, y, ..., name = "attributes")

Arguments

x object of class stars
f the name or index of the dimension to split; by default the last dimension
drop ignored
... if defined, the first unnamed argument is used for dimension values, if not de-
defined, attribute names are used for dimension values
y needs to be missing
name name for the new dimension

Details

split.stars works on the first attribute, and will give an error when more than one attribute is present

Value

merge merges attributes of a stars object into a new dimension; split splits a dimension over attributes

ops_stars S3 Ops Group Generic Functions for stars objects

Description

Ops functions for stars objects, including comparison, product and divide, add, subtract
Usage

```r
## S3 method for class 'stars'
Ops(e1, e2)

## S3 method for class 'stars'
Math(x, ...)

## S3 method for class 'stars_proxy'
Ops(e1, e2)

## S3 method for class 'stars_proxy'
Math(x, ...)
```

Arguments

- `e1`: object of class `stars`
- `e2`: object of class `stars`
- `x`: object of class `stars`
- `...`: parameters passed on to the Math functions

Details

If `e1` or `e2` is a numeric vector, or `e2` has less or smaller dimensions than `e1`, then `e2` is recycled such that it fits `e1`, using usual R array recycling rules. The user needs to make sure this is sensible; it may be needed to use `aperm` to permutate dimensions first.

Value

Object of class `stars`

Examples

```r
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x * x
x / x
x + x
x + 10
all.equal(x * 10, 10 * x)
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
a = sqrt(x)
b = log(x, base = 10)
```
plot stars object, with subplots for each level of first non-spatial dimension, and customization of legend key

Usage

```r
## S3 method for class 'stars'
plot(
x,
y,
..., 
join_zlim = TRUE,
main = make_label(x, 1),
axes = FALSE,
downsampling = TRUE,
nbreaks = 11,
breaks = "quantile",
col = grey(1:(nbreaks - 1)/nbreaks),
key.pos = get_key_pos(x, ...),
key.width = lcm(1.8),
key.length = 0.618,
reset = TRUE,
box.col = grey(0.8),
center.time = FALSE,
hook = NULL,
mfrow = NULL
)
```

```r
## S3 method for class 'stars'
image(
x,
..., 
band = 1,
attr = 1,
asp = NULL,
rgb = NULL,
maxColorValue = ifelse(inherits(rgb, "data.frame"), 255, max(x[[attr]], na.rm = TRUE)),
xlab = if (!axes) "" else names(d)[1],
ylab = if (!axes) "" else names(d)[2],
xlim = st_bbox(extent)$xlim,
ylim = st_bbox(extent)$ylim,
```

Description

plot stars object, with subplots for each level of first non-spatial dimension, and customization of legend key
text_values = FALSE,
        text_color = "black",
        axes = FALSE,
        interpolate = FALSE,
        as_points = FALSE,
        key.pos = NULL,
        logz = FALSE,
        key.width = lcm(1.8),
        key.length = 0.618,
        add.geom = NULL,
        border = NA,
        useRaster = isTRUE(dev.capabilities("rasterImage")$rasterImage == "yes"),
        extent = x
    )

## S3 method for class 'stars_proxy'
plot(x, y, ..., downsample = get_downsample(dim(x)))

Arguments

x
    object of class stars

y
    ignored

...  
    further arguments: for plot, passed on to image.stars; for image, passed on
to image.default or rasterImage.

join_zlim
    logical; if TRUE, compute a single, joint zlim (color scale) for all subplots from
    x

main
    character; subplot title prefix; use "" to get only time, use NULL to suppress
    subplot titles

axes
    logical; should axes and box be added to the plot?

downsample
    logical or numeric; if TRUE will try to plot not many more pixels than actually are
    visible, if FALSE no downsampling takes place, if numeric, the downsampling
    rate; see Details.

nbbreaks
    number of color breaks; should be one more than number of colors. If missing
    and col is specified, it is derived from that.

breaks
    actual color breaks, or a method name used for classIntervals.

col
    colors to use for grid cells

key.pos
    integer; side to plot a color key: 1 bottom, 2 left, 3 top, 4 right; set to NULL
to omit key. Ignored if multiple columns are plotted in a single function call.
Default depends on plot size, map aspect, and, if set, parameter asp.

key.width
    amount of space reserved for width of the key (labels); relative or absolute (using
    lcm)

key.length
    amount of space reserved for length of the key (labels); relative or absolute
    (using lcm)

reset
    logical; if FALSE, keep the plot in a mode that allows adding further map ele-
mments; if TRUE restore original mode after plotting; see details.
box_col: color for box around sub-plots; use 0 to suppress plotting of boxes around sub-plots.

center_time: logical; if TRUE, sub-plot titles will show the center of time intervals, otherwise their start.

hook: NULL or function; hook function that will be called on every sub-plot.

mfrow: length-2 integer vector with nrows, ncolumns of a composite plot, to override the default layout.

band: integer; which band (dimension) to plot.

attr: integer; which attribute to plot.

asp: numeric; aspect ratio of image.

rgb: integer; specify three bands to form an rgb composite. Experimental: rgb color table; see Details.

maxColorValue: numeric; passed on to rgb.

xlab: character; x axis label.

ylab: character; y axis label.

xlim: x axis limits.

ylim: y axis limits.

text_values: logical; print values as text on image?

text_color: character; color for printed text values.

interpolate: logical; when using rasterImage (rgb), should pixels be interpolated?

as_points: logical; for curvilinear or sheared grids: parameter passed on to st_as_sf, determining whether raster cells will be plotted as symbols (fast, approximate) or small polygons (slow, exact).

logz: logical; if TRUE, use log10-scale for the attribute variable. In that case, breaks and at need to be given as log10-values; see examples.

add.geom: object of class sf, or list with arguments to plot, that will be added to an image or sub-image.

border: color used for cell borders (only in case x is a curvilinear or rotated/sheared grid).

useRaster: logical; use the rasterImage capabilities of the graphics device?

extent: object which has a st_bbox method; sets the plotting extent.

**Details**

Downsampling: a value for downsample of 0 or 1 causes no downsampling, 2 that every second dimension value is sampled, 3 that every third dimension value is sampled, and so on; can be specified for each dimension.

Use of an rgb color table is experimental; see https://github.com/r-spatial/mapview/issues/208

When plotting a subsetted stars_proxy object, the default value for argument downsample will not be computed correctly, and it and has to be set manually.
predict.stars

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
image(x, col = grey((3:9)/10))
image(x, rgb = c(1,3,5)) # rgb composite

---

predict.stars

Predict values, given a model object, for a stars or stars_proxy object

Description

Predict values, given a model object, for a stars or stars_proxy object

Usage

## S3 method for class 'stars'
predict(object, model, ...)

## S3 method for class 'stars_proxy'
predict(object, model, ...)

Arguments

object object of class 'stars'
model model object of a class that has a predict method; check with 'methods(class = class(object))'
... arguments passed on to this predict method

Details

separate predictors in object need to be separate attributes in object; in case they are e.g. in a band
dimension, use 'split(object)'

---

read_ncdf

Read NetCDF into stars object

Description

Read data from a file (or source) using the NetCDF library directly.
Usage

`read_ncdf(
  .x,  
  ..., 
  var = NULL, 
  ncsb = NULL, 
  curvilinear = character(0), 
  eps = 1e-12, 
  ignore_bounds = FALSE, 
  make_time = TRUE, 
  make_units = TRUE
)

Arguments

.x NetCDF file or source

... ignored

var variable name or names (they must be on matching grids)

ncsub matrix of start, count columns (see Details)

curvilinear length two character named vector with names of variables holding longitude and latitude values for all raster cells. 'stars' attempts to figure out appropriate curvilinear coordinates if they are not supplied.

eps numeric; dimension value increases are considered identical when they differ less than eps

ignore_bounds logical; should bounds values for dimensions, if present, be ignored?

make_time if TRUE (the default), an attempt is made to provide a date-time class from the "time" variable

make_units if TRUE (the default), an attempt is made to set the units property of each variable

Details

The following logic is applied to coordinates. If any coordinate axes have regularly spaced coordinate variables they are reduced to the offset/delta form with 'affine = c(0, 0)', otherwise the values of the coordinates are stored and used to define a rectilinear grid.

If the data has two or more dimensions and the first two are regular they are nominated as the 'raster' for plotting.

If the curvilinear argument is used it specifies the 2D arrays containing coordinate values for the first two dimensions of the data read. It is currently assumed that the coordinates are 2D and that they relate to the first two dimensions in that order.

If var is not set the first set of variables on a shared grid is used.

start and count columns of ncsb must correspond to the variable dimension (nrows) and be valid index using var.get.nc convention (start is 1-based). If the count value is NA then all steps are included. Axis order must match that of the variable/s being read.
**Examples**

```r
f <- system.file("nc/reduced.nc", package = "stars")
read_ncdf(f)
read_ncdf(f, var = c("anom"))
read_ncdf(f, ncsup = cbind(start = c(1, 1, 1), count = c(10, 12, 1, 1)))
```

# precipitation data in a curvilinear NetCDF
```r
prec_file = system.file("nc/test_stageiv_xyt.nc", package = "stars")
prec = read_ncdf(prec_file, curvilinear = c("lon", "lat"), ignore_bounds = TRUE)
```

```r
# plot(prec) ## gives error about unique breaks
# remove NAs, zeros, and give a large number
# of breaks (used for validating in detail)
qu_0_omit = function(x, ..., n = 22) {
  x = units::drop_units(na.omit(x))
  c(0, quantile(x[x > 0], seq(0, 1, length.out = n)))
}
library(dplyr)
prec_slice = slice(prec, index = 17, along = "time")
plot(prec_slice, border = NA, breaks = qu_0_omit(prec_slice[[1]]), reset = FALSE)
nc = sf::read_sf(system.file("gpkg/nc.gpkg", package = "sf"), "nc.gpkg")
plot(st_geometry(nc), add = TRUE, reset = FALSE, col = NA)
```

---

**read_stars**

**read raster/array dataset from file or connection**

**Description**

read raster/array dataset from file or connection

**Usage**

```r
read_stars(
  .x, 
  ...,
  options = character(0),
  driver = character(0),
  sub = TRUE,
  quiet = FALSE,
  NA_value = NA_real_,
  along = NA_integer_,
  RasterIO = list(),
  proxy = is_functions(.x) || (is_big(.x, sub = sub, driver =
    driver, normalize_path = normalize_path, ...)),
  curvilinear = character(0),
  normalize_path = TRUE,
  RAT = character(0),
```
tolerance = 1e-10

is_big(x, ..., sub = sub, n_proxy = options("stars.n_proxy")[[1]] || 1e+08)

Arguments

\texttt{x} \quad \text{character vector with name(s) of file(s) or data source(s) to be read, or a function that returns such a vector}

\texttt{...} \quad \text{passed on to \texttt{st_as_stars} if \texttt{curvilinear} was set}

\texttt{options} \quad \text{character; opening options}

\texttt{driver} \quad \text{character; driver to use for opening file. To override fixing for subdatasets and autodetect them as well, use \texttt{NULL}.}

\texttt{sub} \quad \text{character, integer or logical; name, index or indicator of sub-dataset(s) to be read}

\texttt{quiet} \quad \text{logical; print progress output?}

\texttt{NA\_value} \quad \text{numeric value to be used for conversion into NA values; by default this is read from the input file}

\texttt{along} \quad \text{length-one character or integer, or list; determines how several arrays are combined, see Details.}

\texttt{RasterIO} \quad \text{list with named parameters for GDAL’s RasterIO, to further control the extent, resolution and bands to be read from the data source; see details.}

\texttt{proxy} \quad \text{logical; if TRUE, an object of class \texttt{stars_proxy} is read which contains array metadata only; if FALSE the full array data is read in memory. Always FALSE for curvilinear grids. If not set, defaults to TRUE when the number of cells to be read is larger than \texttt{options(stars.n_proxy)}, or to 1e8 if that option was not set.}

\texttt{curvilinear} \quad \text{length two character vector with names of subdatasets holding longitude and latitude values for all raster cells, or named length 2 list holding longitude and latitude matrices; the names of this list should correspond to raster dimensions referred to}

\texttt{normalize\_path} \quad \text{logical; if FALSE, suppress a call to \texttt{normalizePath} on \texttt{x}}

\texttt{RAT} \quad \text{character; raster attribute table column name to use as factor levels}

\texttt{tolerance} \quad \text{numeric; passed on to \texttt{all.equal} for comparing dimension parameters.}

\texttt{x} \quad \text{object to be read with \texttt{read_stars}}

\texttt{n\_proxy} \quad \text{integer; number of cells above which \texttt{x} will be read as \texttt{stars_proxy} object, i.e. not as in-memory arrays but left on disk}

Details

In case \texttt{x} contains multiple files, they will all be read and combined with \texttt{c.stars}. Along which dimension, or how should objects be merged? If \texttt{along} is set to \texttt{NA} it will merge arrays as new attributes if all objects have identical dimensions, or else try to merge along time if a dimension called \texttt{time} indicates different time stamps. A single name (or positive value) for \texttt{along} will merge along that dimension, or create a new one if it does not already exist. If the arrays should be arranged along one of more dimensions with values (e.g. time stamps), a named list can passed to \texttt{along} to specify them; see example.
read_stars

RasterIO is a list with zero or more of the following named arguments: nXOff, nYOff (both 1-based: the first row/col has offset value 1), nXSize, nYSize, nBufXSize, nBufYSIZE, bands, coderesample. see https://www.gdal.org/classGDALDataset.html for their meaning; bands is an integer vector containing the band numbers to be read (1-based: first band is 1) Note that if nBufXSize or nBufYSize are specified for downsampling an image, resulting in an adjusted geotransform. resample reflects the resampling method and has to be one of: "nearest_neighbour" (the default), "bilinear", "cubic", "cubic_spline", "lanczos", "average", "mode", or "Gauss".

Value

object of class stars

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
(x1 = read_stars(tif))
(x2 = read_stars(c(tif, tif)))
(x3 = read_stars(c(tif, tif), along = "band"))
(x4 = read_stars(c(tif, tif), along = "new_dimensions")) # create 4-dimensional array
x1o = read_stars(tif, options = "OVERVIEW_LEVEL=1")
t1 = as.Date("2018-07-31")
# along is a named list indicating two dimensions:
read_stars(c(tif, tif, tif, tif), along = list(foo = c("bar1", "bar2"), time = c(t1, t1+2)))

m = matrix(1:120, nrow = 12, ncol = 10)
dim(m) = c(x = 10, y = 12) # named dim
st = st_as_stars(m)
attr(st, "dimensions")$y$delta = -1
attr(st, "dimensions")$y$offset = 12
st
tmp = tempfile(fileext = ".tif")
write_stars(st, tmp)
(red <- read_stars(tmp, RasterIO = list(nXOff = 1, nYOff = 1, nXSize = 10, nYSize = 12,
nBufXSize = 2, nBufYSIZE = 2))[[1]]
(red <- read_stars(tmp, RasterIO = list(nXOff = 1, nYOff = 1, nXSize = 10, nYSize = 12,
nBufXSize = 2, nBufYSIZE = 2)))
red[[1]] # cell values of subsample grid:
## Not run:
plot(st, reset = FALSE, axes = TRUE, ylim = c(-.1,12.1), xlim = c(-.1,10.1),
      main = "nBufXSize & nBufYSize demo", text_values = TRUE)
plot(st_as_sfc(red, as_points = TRUE), add = TRUE, col = 'red', pch = 16)
plot(st_as_sfc(st_as_stars(st), as_points = FALSE), add = TRUE, border = 'grey')
plot(st_as_sfc(red, as_points = FALSE), add = TRUE, border = 'green', lwd = 2)

## End(Not run)
file.remove(tmp)
redimension

redimension array, or collapse attributes into a new dimension

Description

redimension array, or collapse attributes into a new dimension

Usage

st_redimension(x, new_dims, along, ...)

## S3 method for class 'stars'
st_redimension(
  x,
  new_dims = st_dimensions(x),
  along = list(new_dim = names(x)),
  ...
)

## S3 method for class 'stars_proxy'
st_redimension(
  x,
  new_dims = st_dimensions(x),
  along = list(new_dim = names(x)),
  ...
)

Arguments

x          object of class stars
new_dims   target dimensions: either a `dimensions` object or an integer vector with the
dimensions’ sizes
along      named list with new dimension name and values
...        ignored

stars_sentinel2

Sentinel-2 sample tile

Description

Sentinel-2 sample tile, downloaded from https://scihub.copernicus.eu/ reads the four 10-m bands:
B2 (490 nm), B3 (560 nm), B4 (665 nm) and B8 (842 nm)
stars_subset

Usage

stars_sentinel2

Format

An object of class stars_proxy (inherits from stars) of dimension 10980 x 10980 x 4.

Description

subset stars objects

Usage

## S3 method for class 'stars'
x[i = TRUE, ..., drop = FALSE, crop = !is_curvilinear(x)]

## S3 replacement method for class 'stars'
x[i] <- value

st_flip(x, which = 1)

Arguments

x object of class stars

i first selector: integer, logical or character vector indicating attributes to select, or object of class sf or sfc used as spatial selector; see details

... further (logical or integer vector) selectors, matched by order, to select on individual dimensions

drop logical; if TRUE, degenerate dimensions (with only one value) are dropped

crop logical; if TRUE and parameter i is a spatial geometry (sf or sfc) object, the extent (bounding box) of the result is cropped to match the extent of i using st_crop. Cropping curvilinear grids is not supported.

value array of dimensions equal to those in x, or a vector or value that will be recycled to such an array

which character or integer; dimension(s) to be flipped

Details

if i is an object of class sf, sfc or bbox, the spatial subset covering this geometry is selected, possibly followed by cropping the extent. Array values for which the cell centre is not inside the geometry are assigned NA.

in an assignment (or replacement form, [<-), argument i needs to be a stars object with dimensions identical to x, and value will be recycled to the dimensions of the arrays in x.
Value

st_flip flips (reverts) the array values along the chosen dimension without(s) changing the dimension properties

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x[,,1:3]  # select bands
x[,1:100,100:200,]  # select x and y by range
x["L7_ETMs.tif"]  # select attribute
xy = structure(list(x = c(293253.999046018, 296400.196497684), y = c(9113801.64775462, 9111328.49619133)), .Names = c("x", "y"))
pts = st_as_sf(data.frame(do.call(cbind, xy)), coords = c("x", "y"), crs = st_crs(x))
image(x, axes = TRUE)
plot(st_as_sfc(st_bbox(pts)), col = NA, add = TRUE)
bb = st_bbox(pts)
(xx = x[bb])
image(xx)
plot(st_as_sfc(bb), add = TRUE, col = NA)
image(x)
pt = st_point(c(x = 290462.103109179, y = 9114202.32594085))
buf = st_buffer(st_sfc(pt, crs = st_crs(x)), 1500)
plot(buf, add = TRUE)
buf = st_sfc(st_polygon(list(st_buffer(pt, 1500)[[1]], st_buffer(pt, 1000)[[1]])),
crs = st_crs(x))
image(x[buf])
plot(buf, add = TRUE, col = NA)
image(x[buf, crop=FALSE])
plot(buf, add = TRUE, col = NA)
lc = read_stars(system.file("tif/lc.tif", package = "stars"))
x = c(orig = lc,
    flip_x = st_flip(lc, "x"),
    flip_y = st_flip(lc, "y"),
    flip_xy = st_flip(lc, c("x", "y")),
    along = 3)
plot(x)

---

st_apply

---

Description

st_apply apply a function to one or more array dimensions: aggregate over space, time, or something else
Usage

```r
## S3 method for class 'stars'
st_apply(
  X,
  MARGIN,
  FUN,
  ..., 
  CLUSTER = NULL, 
  PROGRESS = FALSE, 
  FUTURE = FALSE, 
  rename = TRUE, 
  .fname, 
  single_arg = has_single_arg(FUN, list(...)) || can_single_arg(FUN)
)
```

Arguments

- **X**: object of class *stars*
- **MARGIN**: see `apply`; index number(s) or name(s) of the dimensions over which `FUN` will be applied
- **FUN**: see `apply` and see Details.
- **...**: arguments passed on to `FUN`
- **CLUSTER**: cluster to use for parallel apply; see `makeCluster`
- **PROGRESS**: logical; if TRUE, use `pbapply::pbapply` to show progress bar
- **FUTURE**: logical; if TRUE, use `future.apply::future_apply`
- **rename**: logical; if TRUE and `X` has only one attribute and `FUN` is a simple function name, rename the attribute of the returned object to the function name
- **.fname**: function name for the new attribute name (if one or more dimensions are reduced) or the new dimension (if a new dimension is created); if missing, the name of `FUN` is used
- **single_arg**: logical; if TRUE, `FUN` takes a single argument (like `fn_ndvi1` below), if FALSE `FUN` takes multiple arguments (like `fn_ndvi2` below).

Details

`FUN` is a function which either operates on a single object, which will be the data of each iteration step over dimensions `MARGIN`, or a function that has as many arguments as there are elements in such an object. See the NDVI examples below. The second form can be VERY much faster e.g. when a trivial function is not being called for every pixel, but only once (example).

The heuristics for the default of `single_arg` work often, but not always; try setting this to the right value when `st_apply` gives an error.

Value

object of class *stars* with accordingly reduced number of dimensions; in case `FUN` returns more than one value, a new dimension is created carrying the name of the function used; see the examples.
Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
st_apply(x, 1:2, mean) # mean band value for each pixel
st_apply(x, c("x", "y"), mean) # equivalent to the above
st_apply(x, 3, mean) # mean of all pixels for each band
st_apply(x, "band", mean) # equivalent to the above
st_apply(x, 1:2, range) # min and max band value for each pixel
fn_ndvi1 = function(x) (x[4]-x[3])/(x[4]+x[3]) # ONE argument: will be called for each pixel
fn_ndvi2 = function(red, nir) (nir-red)/(nir+red) # n arguments: will be called only once
ndvi1 = st_apply(x, 1:2, fn_ndvi1)
ndvi2 = st_apply(x[,3:4], 1:2, fn_ndvi2) # note that we select bands 3 and 4 in the first argument
all.equal(ndvi1, ndvi2)
# compute the (spatial) variance of each band; https://github.com/r-spatial/stars/issues/430
st_apply(x, 3, function(x) var(as.vector(x))) # as.vector is required!
# to get a progress bar also in non-interactive mode, specify:
if (require(pbapply)) { # install it, if FALSE
  pboptions(type = "timer")
}

st_as_sf

Convert stars object into an sf object

Description

Convert stars object into an sf object

Usage

## S3 method for class 'stars'
st_as_sfc(x, ..., as_points, which = seq_len(prod(dim(x)[1:2])))

## S3 method for class 'stars'
st_as_sf(
x,
  ..., 
as_points = FALSE,
  merge = FALSE,
  na.rm = TRUE,
  use_integer = is.logical(x[[1]]) || is.integer(x[[1]]),
  long = FALSE,
  connect8 = FALSE
)

## S3 method for class 'stars_proxy'
st_as_sf(x, ..., downsample = 0)
Arguments

- x: object of class stars
- ...: ignored
- as_points: logical; should cells be converted to points or to polygons? See details.
- which: linear index of cells to keep (this argument is not recommended to be used)
- merge: logical; if TRUE, cells with identical values are merged (using GDAL_Polygonize or GDAL_FPolygonize); if FALSE, a polygon for each raster cell is returned; see details
- na.rm: logical; should missing valued cells be removed, or also be converted to features?
- use_integer: (relevant only if merge is TRUE): if TRUE, before polygonizing values are rounded to 32-bits signed integer values (GDALPolygonize), otherwise they are converted to 32-bit floating point values (GDALFPolygonize).
- long: logical; if TRUE, return a long table form sf, with geometries and other dimensions recycled
- connect8: logical; if TRUE, use 8 connectedness. Otherwise the 4 connectedness algorithm will be applied.
- downsample: see st_as_stars

Details

If merge is TRUE, only the first attribute is converted into an sf object. If na.rm is FALSE, areas with NA values are also written out as polygons. Note that the resulting polygons are typically invalid, and use st_make_valid to create valid polygons out of them.

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
x = x[,1:100,1:100,6] # subset of a band with lower values in it
x[[1]][x[[1]] < 30] = NA # set lower values to NA
x[[1]] = x[[1]] < 100 # make the rest binary
x
(p = st_as_sf(x)) # removes NA areas
(p = st_as_sf(x[,,1], merge = TRUE)) # glues polygons together
all(st_is_valid(p)) # not all valid, see details
plot(p, axes = TRUE)
(p = st_as_sf(x, na.rm = FALSE, merge = TRUE)) # includes polygons with NA values
plot(p, axes = TRUE)
st_as_stars

**Description**

convert objects into a stars object

**Usage**

```r
st_as_stars(.x, ...)
```

```r
## S3 method for class 'list'
st_as_stars(.x, ..., dimensions = NULL)
```

```r
## Default S3 method:
st_as_stars(.x = NULL, ..., raster = NULL)
```

```r
## S3 method for class 'stars'
st_as_stars(.x, ..., curvilinear = NULL, crs = st_crs(4326))
```

```r
## S3 method for class 'bbox'
st_as_stars(.x, ..., nx, ny, dx = dy, dy = dx, xlim = .x[c("xmin", "xmax")], ylim = .x[c("ymin", "ymax")], values = 0, n = 64800, pretty = FALSE, inside = FALSE, nz)
```

```r
## S3 method for class 'sf'
st_as_stars(.x, ..., name = attr(.x, "sf_column"))
```

```r
## S3 method for class 'Raster'
st_as_stars(.x, ..., att = 1, ignore_file = FALSE)
```

```r
## S3 method for class 'ncdfgeom'
st_as_stars(.x, ..., sf_geometry = NA)
```

```r
## S3 method for class 'stars_proxy'
```
st_as_stars

st_as_stars(
  .x,
  ..., 
  downsample = 0,
  url = attr(.x, "url"),
  envir = parent.frame()
)

## S3 method for class 'data.frame'
st_as_stars(
  .x,
  ..., 
  dims = coords,
  xy = dims[1:2],
  y_decreasing = TRUE,
  coords = 1:2
)

## S3 method for class 'xts'
st_as_stars(.x, ..., dimensions)

## S3 method for class 'OpenStreetMap'
st_as_stars(.x, ..., as_col = FALSE)

Arguments

.x  object to convert
...
in case .x is of class bbox, arguments passed on to pretty
dimensions object of class dimensions
raster character; the names of the dimensions that denote raster dimensions
curvilinear only for creating curvilinear grids: named length 2 list holding longitude and
latitude matrices; the names of this list should correspond to raster dimensions
referred to

crs object of class crs with the coordinate reference system of the values in curvilinear; see
details

nx integer; number of cells in x direction; see details
ny integer; number of cells in y direction; see details
dx numeric; cell size in x direction; see details
dy numeric; cell size in y direction; see details
xlim length 2 numeric vector with extent (min, max) in x direction
ylim length 2 numeric vector with extent (min, max) in y direction
values value(s) to populate the raster values with
n the (approximate) target number of grid cells

pretty logical; should cell coordinates have pretty values?
inside logical; should all cells entirely fall inside the bbox, potentially not covering it completely?
nz integer; number of cells in z direction; if missing no z-dimension is created.
name character; name for the geometry dimensions
att see factorValues; column in the RasterLayer’s attribute table
ignore_file logical; if TRUE, ignore the Raster object file name
sf_geometry sf data.frame with geometry and attributes to be added to stars object. Must have same number of rows as timeseries instances.
downsampel integer: if larger than 0, downsample with this rate (number of pixels to skip in every row/column); if length 2, specifies downsampling rate in x and y.
url character; URL of the stars endpoint where the data reside
envir environment to resolve objects in
dims the column names or indices that form the cube dimensions
xy the x and y raster dimension names or indices; only takes effect after dims has been specified
y_decreasing logical; if TRUE, (numeric) y values get a negative delta (decrease with increasing index)
coords same as dims, for symmetry with st_as_sf
as_col logical; return rgb numbers (FALSE) or (character) color values (TRUE)?

Details

if curvilinear is a stars object with longitude and latitude values, its coordinate reference system is typically not that of the latitude and longitude values.

For the bbox method: if pretty is TRUE, raster cells may extend the coordinate range of .x on all sides. If in addition to nx and ny, dx and dy are also missing, these are set to a single value computed as sqrt((diff(xlim)*diff(ylim))/n). If nx and ny are missing, they are computed as the ceiling of the ratio of the (x or y) range divided by (dx or dy), unless inside is TRUE, in which case ceiling is replaced by floor. Positive dy will be made negative. Further named arguments (…) are passed on to pretty.

For the ncdfgeom method: objects are point-timeseries with optional line or polygon geometry for each timeseries specified with the sf_geometry parameter. See ncdfgeom for more about this NetCDF-based format for geometry and timeseries.

for the xts methods, if dimensions are provided, time has to be the first dimension.

Examples

data(Produc, package = "plm")
st_as_stars(Produc, y_decreasing = FALSE)
st_contour

Compute or plot contour lines or sets

Description

Compute contour lines or sets

Usage

st_contour(
  x,
  na.rm = TRUE,
  contour_lines = FALSE,
  breaks = classInt::classIntervals(na.omit(as.vector(x[[1]])))$brks
)

Arguments

x object of class stars
na.rm logical; should missing valued cells be removed, or also be converted to features?
contour_lines logical; if FALSE, polygons are returned (contour sets), otherwise contour lines
breaks numerical; values at which to "draw" contour levels

Details

this function requires GDAL >= 2.4.0

See Also

for polygonizing rasters following grid boundaries, see st_as_sf with arguments as_points=FALSE and merge=TRUE; contour plots contour lines using R’s native algorithm (which also plots contour levels)

st_coordinates

retrieve coordinates for raster or vector cube cells

Description

retrieve coordinates for raster or vector cube cells
Usage

```r
## S3 method for class 'stars'
st_coordinates(x, ..., add_max = FALSE, center = TRUE)
```

```r
## S3 method for class 'stars'
as.data.frame(x, ..., add_max = FALSE, center = NA)
```

```r
as_tibble.stars(.x, ..., add_max = FALSE, center = NA)
```

Arguments

- `x`: object of class `stars`
- `...`: ignored
- `add_max`: logical; if TRUE, dimensions are given with a min (x) and max (x_max) value
- `center`: logical; (only if `add_max` is FALSE): should grid cell center coordinates be returned (TRUE) or offset values (FALSE)? center can be a named logical vector or list to specify values for each dimension.
- `.x`: object to be converted to a tibble

---

### st_crop

**crop a stars object**

Description

crop a stars object

Usage

```r
## S3 method for class 'stars_proxy'
st_crop(
  x,
  y,
  ...,
  crop = TRUE,
  epsilon = sqrt(.Machine$double.eps),
  collect = TRUE
)
```

```r
## S3 method for class 'stars'
st_crop(
  x,
  y,
  ...,
  crop = TRUE,
  epsilon = sqrt(.Machine$double.eps),
  as_points = all(st_dimension(y) == 2, na.rm = TRUE)
)
st_crop

Arguments

x object of class stars
y object of class sf, sfc or bbox; see Details below.
... ignored
crop logical; if TRUE, the spatial extent of the returned object is cropped to still cover obj, if FALSE, the extent remains the same but cells outside y are given NA values.
epsilon numeric; factor to shrink the bounding box of y towards its center before cropping.
collect logical; if TRUE, repeat cropping on stars object, i.e. after data has been read
as_points logical; only relevant if y is of class sf or sfc: if FALSE, treat x as a set of points, else as a set of small polygons. Default: TRUE if y is two-dimensional, else FALSE; see Details

Details

for raster x, st_crop selects cells that intersect with y. For intersection, are raster cells interpreted as points or as small polygons? If y is of class stars, x raster cells are interpreted as points; if y is of class bbox, x cells are interpreted as cells (small polygons). Otherwise, if as_points is not given, cells are interpreted as points if y has a two-dimensional geometry.

Examples

l7 = read_stars(system.file("tif/L7_ETMs.tif", package = "stars"))
d = st_dimensions(l7)

# area around cells 3:10 (x) and 4:11 (y):
offset = c(d["x"]$offset, d["y"]$offset)
res = c(d["x"]$delta, d["y"]$delta)
bb = st_bbox(c(xmin = offset[1] + 2 * res[1],
ymin = offset[2] + 11 * res[2],
xmax = offset[1] + 10 * res[1],
ymax = offset[2] + 3 * res[2]), crs = st_crs(l7))
l7[bb]

plot(l7[,...,1,1], reset = FALSE)
image(l7[bb,...,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

# slightly smaller bbox:
bb = st_bbox(c(xmin = offset[1] + 2.1 * res[1],
          ymin = offset[2] + 10.9 * res[2],
          xmax = offset[1] + 9.9 * res[1],
l7[bb]

plot(l7[,...,1,1], reset = FALSE)
image(l7[bb,...,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)
# slightly larger bbox:
bb = st_bbox(c(xmin = offset[1] + 1.9 * res[1],
xmax = offset[1] + 10.1 * res[1],
ymax = offset[2] + 2.9 * res[2]), crs = st_crs(l7))
l7[bb]

plot(l7[,1:13,1:13,1], reset = FALSE)
image(l7[bb,,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

# half a cell size larger bbox:
bb = st_bbox(c(xmin = offset[1] + 1.49 * res[1],
ymin = offset[2] + 11.51 * res[2],
xmax = offset[1] + 10.51 * res[1],
ymax = offset[2] + 2.49 * res[2]), crs = st_crs(l7))
l7[bb]

plot(l7[,1:13,1:13,1], reset = FALSE)
image(l7[bb,,,1], add = TRUE, col = sf.colors())
plot(st_as_sfc(bb), add = TRUE, border = 'green', lwd = 2)

---

**st_dimensions**

get dimensions from stars object

**Description**

get dimensions from stars object

**Usage**

```r
st_dimensions(.x, ...)  
```

### S3 method for class 'stars'

```r
st_dimensions(.x, ...)  
```

`st_dimensions(x) <- value`

### S3 replacement method for class 'stars'

```r
st_dimensions(x) <- value`
```

### S3 replacement method for class 'list'

```r
st_dimensions(x) <- value`
```

### S3 method for class 'array'

```r
st_dimensions(x, ...)`
```

### Default S3 method:

```r
st_dimensions`
```
st_dimensions

.x,
...,  .raster,
affine = c(0, 0),
    cell_midpoints = FALSE,
    point = FALSE
)

st_set_dimensions(
   .x,  which,
    values = NULL,
    point = NULL,
    names = NULL,
    xy,
      ...
)

st_get_dimension_values(.x, which, ..., where = NA, max = FALSE, center = NA)

Arguments

.x
  object to retrieve dimensions information from
...
  further arguments
x
  object of class dimensions
value
  new object of class dimensions, with matching dimensions
.raster
  length 2 character array with names (if any) of the raster dimensions
affine
  numeric; specify parameters of the affine transformation
cell_midpoints
  logical; if TRUE AND the dimension values are strictly regular, the values are interpreted as the cell midpoint values rather than the cell offset values when calculating offset (i.e., the half-cell-size correction is applied); can have a value for each dimension, or else is recycled
point
  logical; does the pixel value (measure) refer to a point (location) value or to an pixel (area) summary value?
which
  integer or character; index or name of the dimension to be changed
values
  values for this dimension (e.g. sfc list-column), or length-1 dimensions object
names
  character; vector with new names for all dimensions, or with the single new name for the dimension indicated by which
xy
  length-2 character vector; (new) names for the x and y raster dimensions
where
  character, one of 'start', 'center' or 'end'. Set to NA (default) to ignore and use max and center explicitly. This argument provides a convenient alternative to setting max and center.
max
  logical; if TRUE return the end, rather than the beginning of an interval
center
  logical; if TRUE return the center of an interval; if NA return the center for raster dimensions, and the start of intervals in other cases
Details

dimensions can be specified in two ways. The simplest is to pass a vector with numeric values for a numeric dimension, or character values for a categorical dimension. Parameter cell_midpoints is used to specify whether numeric values refer to the offset (start) of a dimension interval (default), or to the center; the center case is only available for regular dimensions. For rectilinear numeric dimensions, one can specify either a vector with cell borders (start values), or a data.frame with two columns named "start" and "end", with the respective interval start and end values. In the first case, the end values are computed from the start values by assuming the last two intervals have equal width.

Value

the dimensions attribute of x, of class dimensions

Examples

```r
x = read_stars(system.file("tif/L7_ETMs.tif", package = "stars"))
# Landsat 7 ETM+ band semantics: https://landsat.gsfc.nasa.gov/the-enhanced-thematic-mapper-plus/
# set bands to values 1,2,3,4,5,7:
(x1 = st_set_dimensions(x, "band", values = c(1,2,3,4,5,7), names = "band_number", point = TRUE))
# set band values as bandwidth
rbind(c(0.45,0.515), c(0.525,0.605), c(0.63,0.69), c(0.775,0.90), c(1.55,1.75), c(2.08,2.35)) %>%
units::set_units("um") -> bw # or: units::set_units(\micro m) -> bw
# set bandwidth midpoint:
(x2 = st_set_dimensions(x, "band", values = 0.5 * (bw[,1]+bw[,2]),
   names = "bandwidth_midpoint", point = TRUE))
# set bandwidth intervals:
(x3 = st_set_dimensions(x, "band", values = make_intervals(bw), names = "bandwidth"))
```

```r
m = matrix(1:20, nrow = 5, ncol = 4)
dim(m) = c(x = 5, y = 4) # named dim
(s = st_as_stars(m))
st_get_dimension_values(s, 'x', where = "start")
st_get_dimension_values(s, 'x', center = FALSE)
st_get_dimension_values(s, 'x', where = "center")
st_get_dimension_values(s, 'x', center = TRUE)
st_get_dimension_values(s, 'x', where = "end")
st_get_dimension_values(s, 'x', max = TRUE)
```

---

**st_dim_to_attr**

create an array with dimension values

Description

create an array with dimension values

Usage

```r
st_dim_to_attr(x, which = seq_along(dim(x)))
```
st_extract

Arguments

x object of class stars
which integer; indices of the dimensions to address (default: all)

Value

stars object with dimension values as attributes

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x1 = read_stars(tif)
(x = st_dim_to_attr(x1))
plot(x)
(x = st_dim_to_attr(x1, 2:3))
plot(x)
(x= st_dim_to_attr(x1, 3))
plot(x)

st_extract

Extract cell values at point locations

Description

Extract cell values at point locations

Usage

st_extract(x, ...)

## S3 method for class 'stars'
st_extract(
x, at, ...
, bilinear = FALSE,
time_column = attr(at, "time_column") %||% attr(at, "time_col"),
interpolate_time = bilinear,
FUN = mean
)

Arguments

x object of class stars or stars_proxy
...
passed on to aggregate.stars when geometries are not exclusively POINT geometries
at object of class sf or sfc with geometries, where to extract x
st_intersects.stars

bilinear logical; use bilinear interpolation rather than nearest neighbour?
time_column character or integer; name or index of a column with time or date values that will be matched to values of the dimension "time" in x, after which this dimension is reduced. This is useful to extract data cube values along a trajectory; see https://github.com/r-spatial/stars/issues/352.
interpolate_time logical; should time be interpolated? if FALSE, time instances are matched using the coinciding or the last preceding time in the data cube.
FUN function used to aggregate pixel values when geometries of at intersect with more than one pixel

Details

points outside the raster are returned as NA values.

Value

if x has more dimensions than only x and y (raster), an object of class stars with POINT geometries replacing x and y raster dimensions; otherwise an object of sf with extracted values.

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
r = read_stars(tif)
pnt = st_sample(st_as_sfc(st_bbox(r)), 10)
st_extract(r, pnt)
st_extract(r[,,,1], pnt)

---

st_intersects.stars spatial intersect predicate for stars and sfc object

Description

spatial intersect predicate for stars and sfc object

Usage

## S3 method for class 'stars'
st_intersects(x, y, sparse = TRUE, ..., as_points = NA, transpose = FALSE)

Arguments

x object of class stars
y object that has an 'st_geometry' method: of class 'sf' or 'sfc', or 'stars' object with an 'sfc' dimension
st_join.stars

Spatially join a stars and an 'sf' object

## S3 method for class 'stars'
st_join(
x, y, join = st_intersects, ...
... what = "left1", as_points = NA,
warn = TRUE
)

Arguments

- **x** object of class stars
- **y** object of class sf, or one that can be coerced into that by st_as_sf
- **join** the join function, which should return an sgbp object; see details
- **...** arguments that will be passed on to the join function
- **what** "left1", "right" or "inner"; see details
- **as_points** logical; controls whether grid cells in x will be treated as points, or as cell areas; the st_intersects.stars method by default will derive this from x's metadata, or else assume areas.
- **warn** logical; if TRUE, warn on 1-to-many matches when what is "left1"

Details

Curvilinear grids are always converted to polygons, so points on grid boundaries may intersect with two cells touched; for other grids each cell boundary or corner belongs only to one cell.

Value

'sgbp' object if sparse = TRUE, logical matrix otherwise
Details

When there is more than one match to a single x value, the first matching record from y is taken (and if warn is TRUE a warning is raised). If what is "inner", an object of class sf with all matching records of x and y.

Value

If what is "left1", an object of class stars with the (first) value of y at spatial instances of x

---

**st_mosaic**  
*build mosaic (composite) of several spatially disjoint stars objects*

Description

build mosaic (composite) of several spatially disjoint stars objects

Usage

```r
st_mosaic(.x, ...)

## S3 method for class 'stars'
st_mosaic(
  .x,
  ...,  
  dst = tempfile(fileext = file_ext),
  options = c("-vrtnodata", "-9999", "-srcnodata", "nan"),
  file_ext = ".tif"
)

## S3 method for class 'character'
st_mosaic(
  .x,
  ...,  
  dst = tempfile(fileext = file_ext),
  options = c("-vrtnodata", "-9999"),
  file_ext = ".tif"
)

## S3 method for class 'stars_proxy'
st_mosaic(
  .x,
  ...,  
  dst = tempfile(fileext = file_ext),
  options = c("-vrtnodata", "-9999"),
  file_ext = ".tif"
)
```
Arguments

- \texttt{x} \hspace{1cm} \text{object of class stars, or character vector with input dataset names}
- \texttt{...} \hspace{1cm} \text{further input stars objects}
- \texttt{dst} \hspace{1cm} \text{character; destination file name}
- \texttt{options} \hspace{1cm} \text{character; options to the gdalbuildvrt command}
- \texttt{file_ext} \hspace{1cm} \text{character; file extension, determining the format used to write to (".tif" implies GeoTIFF)}

Details

the gdal function buildvrt builds a mosaic of input images; these input images can be multi-band, but not higher-dimensional data cubes or stars objects with multiple attributes
uses \texttt{gdal_utils} to internally call \texttt{buildvrt}; no executables external to R are called.

Value

the stars method returns a stars object with the composite of the input; the character method returns the file name of the file with the mosaic; see also the GDAL documentation of \texttt{gdalbuildvrt}

Examples

\begin{verbatim}
x = read_stars(system.file("tif/L7_ETMs.tif", package = "stars"))
x1 = x[,100:200,100:200,]
x2 = x[,150:300,150:300,]
plot(st_mosaic(x1, x2))
\end{verbatim}
**st_raster_type**

get the raster type (if any) of a stars object

### Description

get the raster type (if any) of a stars object

### Usage

```r
st_raster_type(x)
```

### Arguments

- `x` object of class stars
Value

one of NA (if the object does not have raster dimensions), "curvilinear", "rectilinear", "affine", or "regular"

Examples

tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
st_raster_type(x)

---

**st_rgb**

reduce dimension to rgb (alpha) hex values

Description

reduce dimension to rgb (alpha) hex values

Usage

```r
st_rgb(
  x,
  dimension = 3,
  use_alpha = dim(x)[dimension] == 4,
  maxColorValue = 255L,
  probs = c(0, 1),
  stretch = FALSE
)
```

Arguments

- `x`: object of class `stars`
- `dimension`: dimension name or number to reduce
- `use_alpha`: logical; if TRUE, the fourth band will be used as alpha values
- `maxColorValue`: integer; maximum value for colors
- `probs`: probability values for quantiles used for stretching
- `stretch`: logical; if TRUE, each band is stretched to 0 ... maxColorValue

Details

the dimension’s bands are mapped to red, green, blue, alpha; if a different ordering is wanted, use `.stars` to reorder a dimension, see examples

See Also

`st_apply`, `rgb`
**Examples**

```r
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
st_rgb(x[,,,3:1])
r = st_rgb(x[,,,c(6,5,4,3)], 3, use_alpha=TRUE) # now R=6,G=5,B=4,alpha=3
if (require(ggplot2)) {
  ggplot() + geom_stars(data = r) + scale_fill_identity()
}
```

**Description**

set bounding box parameters of regular grid

**Usage**

```r
st_set_bbox(x, value, ...)
```

**Arguments**

- `x`: object of class dimensions, stars or stars_proxy
- `value`: object of class bbox
- `...`: ignored

**Value**

object of class stars with a POINT list replaced by x and y raster dimensions. This only works when the points are distributed over a regular or rectilinear grid.
st_transform

transform geometries in stars objects to a new coordinate reference system, without warping

Description

transform geometries in stars objects to a new coordinate reference system, without warping

Usage

## S3 method for class 'stars'
st_transform(x, crs, ...)

## S3 method for class 'stars'
st_transform_proj(x, crs, ...)

Arguments

x object of class stars, with either raster or simple feature geometries
crs object of class crs with target crs
... ignored

details
For simple feature dimensions, st_transform is called, leading to lossless transformation. For gridded spatial data, a curvilinear grid with transformed grid cell (centers) is returned, which is also lossless. To convert this to a regular grid in the new CRS, use st_warp (which is in general lossy).

See Also

st_warp

Examples

geomatrix = system.file("tif/geomatrix.tif", package = "stars")
(x = read_stars(geomatrix))
new = st_crs(4326)
y = st_transform(x, new)
plot(st_transform(st_as_sfc(st_bbox(x)), new), col = NA, border = 'red')
plot(st_as_sfc(y, as_points=FALSE), col = NA, border = 'green', axes = TRUE, add = TRUE)
image(y, col = heat.colors(12), add = TRUE)
plot(st_as_sfc(y, as_points=TRUE), pch=3, cex=.5, col = 'blue', add = TRUE)
plot(st_transform(st_as_sfc(x, as_points=FALSE), new), add = TRUE)
**st_warp**  
*Warp (resample) grids in stars objects to a new grid, possibly in a new coordinate reference system*

**Description**

Warp (resample) grids in stars objects to a new grid, possibly in a new coordinate reference system.

**Usage**

```r
st_warp(
  src,
  dest,
  ..., 
  crs = NA_crs_,
  cellsize = NA_real_,
  segments = 100,
  use_gdal = FALSE,
  options = character(0),
  no_data_value = NA_real_,
  debug = FALSE,
  method = "near"
)
```

**Arguments**

- **src**: object of class `stars` with source raster
- **dest**: object of class `stars` with target raster geometry
- **...**: ignored
- **crs**: coordinate reference system for destination grid, only used when `dest` is missing
- **cellsize**: length 1 or 2 numeric; cellsize in target coordinate reference system units
- **segments**: (total) number of segments for segmentizing the bounding box before transforming to the new crs
- **use_gdal**: logical; if TRUE, use gdalwarp, through `gdal_utils`
- **options**: character vector with options, passed on to gdalwarp
- **no_data_value**: value used by gdalwarp for no_data (NA) when writing to temporary file
- **debug**: logical; if TRUE, do not remove the temporary gdalwarp destination file, and print its name
- **method**: character; see details for options; methods other than near only work when `use_gdal=TRUE`
Details

method should be one of near, bilinear, cubic, cubicspline, lanczos, average, mode, max, min, med, q1 or q3; see https://github.com/r-spatial/stars/issues/109

For gridded spatial data (dimensions x and y), see figure; the existing grid is transformed into a regular grid defined by dest, possibly in a new coordinate reference system. If dest is not specified, but crs is, the procedure used to choose a target grid is similar to that of projectRaster (currently only with method = 'ngb'). This entails: (i) the envelope (bounding box polygon) is transformed into the new crs, possibly after segmentation (red box); (ii) a grid is formed in this new crs, touching the transformed envelope on its East and North side, with (if cellsize is not given) a cellsize similar to the cell size of src, with an extent that at least covers x; (iii) for each cell center of this new grid, the matching grid cell of x is used; if there is no match, an NA value is used.

Examples

```r
gematrix = system.file("tif/geomatrix.tif", package = "stars")
(x = read_stars(geomatrix))
new_crs = st_crs(4326)
y = st_warp(x, crs = new_crs)
plot(st_transform(st_as_sfc(st_bbox(x)), new_crs), col = NA, border = "red")
plot(st_as_sfc(y, as_points=FALSE), col = NA, border = "green", axes = TRUE, add = TRUE)
image(y, add = TRUE, nbreaks = 6)
plot(st_as_sfc(y, as_points=TRUE), pch=3, cex=.5, col = "blue", add = TRUE)
plot(st_transform(st_as_sfc(x, as_points=FALSE), new_crs), add = TRUE)

# warp 0-360 raster to -180-180 raster:
r = read_stars(system.file("nc/reduced.nc", package = "stars"))
r %>% st_set_crs(4326) %>% st_warp(st_as_stars(st_bbox(), dx = 2)) -> s
plot(r, axes = TRUE) # no CRS set, so no degree symbols in labels
plot(s, axes = TRUE)

# downsample raster (90 to 270 m)
r = read_stars(system.file("tif/olinda_dem_utm25s.tif", package = "stars"))
r270 = st_as_stars(st_bbox(r), dx = 270)
r270 = st_warp(r, r270)
```

---

st_xy2sfc

replace x y raster dimensions with simple feature geometry list (points, or polygons = rasterize)

Description

replace x y raster dimensions with simple feature geometry list (points, or polygons = rasterize)

Usage

```r
st_xy2sfc(x, as_points, ..., na.rm = TRUE)
```
Arguments

- `x`: object of class `stars`
- `as_points`: logical; if TRUE, generate points at cell centers, else generate polygons
- `...`: arguments passed on to `st_as_sfc`
- `na.rm`: logical; omit (remove) cells which are entirely missing valued (across other dimensions)?

Value

Object of class `stars` with x and y raster dimensions replaced by a single sfc geometry list column containing either points, or polygons. Adjacent cells with identical values are not merged; see `st_rasterize` for this.

Description

Write stars object to gdal dataset (typically: to file)

Usage

```r
write_stars(obj, dsn, layer, ...)  
## S3 method for class 'stars'
write_stars(
  obj,
  dsn,
  layer = 1,
  ...,  
  driver = detect.driver(dsn),
  options = character(0),
  type = if (is.factor(obj[[1]]) && length(levels(obj[[1]])) < 256) "Byte" else "Float32",
  NA_value = NA_real_,
  update = FALSE,
  normalize_path = TRUE
)

## S3 method for class 'stars_proxy'
write_stars(
  obj,
  dsn,
  layer = 1,
  ...,  
  driver = detect.driver(dsn),
)```
options = character(0),
    type = "Float32",
    NA_value = NA_real_,
    chunk_size = c(dim(obj)[1], floor(2.5e+07/dim(obj)[1]),
        progress = TRUE
    )

detect.driver(filename)

Arguments

obj object of class stars
dsn gdal dataset (file) name
layer attribute name; if missing, the first attribute is written
... passed on to gdal_write
driver driver driver name; see st_drivers
options character vector with dataset creation options, passed on to GDAL
type character; output binary type, one of: Byte for eight bit unsigned integer, UInt16 for sixteen bit unsigned integer, Int16 for sixteen bit signed integer, UInt32 for thirty two bit unsigned integer, Int32 for thirty two bit signed integer, Float32 for thirty two bit floating point, Float64 for sixty four bit floating point.
NA_value non-NA value that should represent R’s NA value in the target raster file; if set to NA, it will be ignored.
update logical; if TRUE, an existing file is being updated
normalize_path logical; see read_stars
chunk_size length two integer vector with the number of pixels (x, y) used in the read/write loop; see details.
progress logical; if TRUE, a progress bar is shown
filename character; used for guessing driver short name based on file extension; see examples

Details

write_stars first creates the target file, then updates it sequentially by writing blocks of chunk_size. In case obj is a multi-file stars_proxy object, all files are written as layers into the output file dsn

Examples

detect.driver("L7_ETMs.tif")
%in%,stars-method    evaluate whether cube values are in a given set

Description

evaluate whether cube values are in a given set

Usage

## S4 method for signature 'stars'
x %in% table

Arguments

x      data cube value
table  values of the set
Index

* datasets
  bcsd_obs, 5
  L7_ETMs, 11
  stars_sentinel2, 22
  [.stars, 43
  [.stars(stars_subset), 23
  [<-.stars(stars_subset), 23
  %in%,stars-method, 50
  
aes, 10
  aggregate(aggregate.stars), 3
  aggregate.stars, 3, 37
  all.equal, 6, 20
  apply, 25
  as, 5
  as.data.frame.stars, 44
  as.data.frame.stars(st_coordinates), 31
  as.tbl_cube.stars(dplyr), 9
  as_tibble.stars(st_coordinates), 31
  
  bcsd_obs, 5
  c.stars, 6, 20
  c.stars_proxy(c.stars), 6
  classIntervals, 15
  coerce,stars,Raster-method(as), 5
  coerce,stars_proxy,Raster-method(as), 5
  contour, 7, 31
  contour.stars, 7
  coord_equal, 10
  coverage_fraction, 4
  cut, 8
  cut.array(cut_stars), 8
  cut.matrix(cut_stars), 8
  cut.POSIXt, 3, 4
  cut.stars(cut_stars), 8
  cut_stars, 8
  
detect.driver(write_stars), 48
  dplyr, 9
  facet_wrap, 10
  factorValues, 30
  filter, 9
  filter.stars(dplyr), 9
  filter.stars_proxy(dplyr), 9
  findInterval, 4
  gdal_utils, 41, 46
  gdal_write, 49
  geom_raster, 10
  geom_sf, 10
  geom_stars, 10
  geom_tile, 10
  image.stars(plot), 14
  is_big(read_stars), 19
  
  L7_ETMs, 11
  make_intervals, 11
  makeCluster, 25
  Math.stars(ops_stars), 12
  Math.stars_proxy(ops_stars), 12
  merge, 12
  mutate.stars(dplyr), 9
  mutate.stars_proxy(dplyr), 9
  normalizePath, 20
  
  Ops.stars(ops_stars), 12
  Ops.stars_proxy(ops_stars), 12
  ops_stars, 12
  
  plot, 14
  predict.stars, 17
  predict.stars_proxy(predict.stars), 17
  pretty, 29
  projectRaster, 47
  pull, 9
  pull.stars(dplyr), 9
  pull.stars_proxy(dplyr), 9

51
rasterImage, 16
read_ncdf, 17
read_stars, 6, 19, 20, 49
redimension, 22
replace_na, 10
replace_na.stars (dplyr), 9
replace_na.stars_proxy (dplyr), 9
rgb, 16, 43
select.stars (dplyr), 9
select.stars_proxy (dplyr), 9
slice.stars (dplyr), 9
slice.stars_proxy (dplyr), 9
split (merge), 12
st_apply, 24, 43
st_as_sf, 3, 16, 26, 30, 31, 39
st_as_sfc.stars (st_as_sf), 26
st_as_stars, 20, 27, 28, 42
st_contour, 7, 31
st_coordinates, 31
st_crop, 23, 32
st_dim_to_attr, 36
st_dimensions, 34
st_dimensions<- (st_dimensions), 34
st_drivers, 49
st_extract, 37
st_flip (stars_subset), 23
st_get_dimension_values
   (st_dimensions), 34
st_intersects.stars, 38, 39
st_join.stars, 39
st_make_valid, 27
st_mosaic, 40
st_raster_type, 42
st_rasterize, 41
st_redimension (redimension), 22
st_rgb, 43
st_set_bbox, 44
st_set_dimensions (st_dimensions), 34
st_sfc2xy, 44
st_transform, 45, 45
st_transform_proj.stars (st_transform), 45
st_warp, 45, 46
st_xy2sfc, 47
stars_sentinel2, 22
stars_subset, 23
theme_stars (geom_stars), 10
transmute.stars (dplyr), 9
transmute.stars_proxy (dplyr), 9
var.get.nc, 18
write_stars, 48