Package 'stars'

December 14, 2025

Title Spatiotemporal Arrays, Raster and Vector Data Cubes

Version 0.7-0

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 https://cran.uni-muenster.de/pebesma/

LazyData true

Depends R (>= 4.1.0), abind, sf (>= 1.0-19)

Imports methods, parallel, classInt (>= 0.4-1), rlang, units

Suggests Cairo, CFtime (>= 1.7.1), OpenStreetMap, RNetCDF (>= 1.8-2), clue, covr, cubble (>= 0.3.0), cubelyr, digest, dplyr (>= 0.7-0), exactextractr, FNN, future.apply, ggforce, ggplot2, ggthemes, gstat, httr, jsonlite, knitr, lwgeom, maps, mapdata, ncdfCF, ncdfgeom, ncmeta (>= 0.0.3), pbapply, plm, randomForest, raster, rmarkdown, sp, spacetime, spatstat (>= 2.0-1), spatstat.geom, starsdata, terra (>= 1.4-22), testthat, tibble, tidyr, tsibble, viridis, xts, zoo

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 7.3.3

Collate 'aggregate.R' 'cf.R' 'cubble.R' 'datasets.R' 'dimensions.R' 'extract.R' 'factors.R' 'geom.R' 'gt.R' 'init.R' 'intervals.R' 'mdim.R' 'mosaic.R' 'ncdf.R' 'ncproxy.R' 'OpenStreetMap.R' 'ops.R' 'plot.R' 'prcomp.R' 'proxy.R' 'rasterize.R' 'raster.R' 'read.R' 'rotate.R' 'sample.R' 'sf.R' 'spacetime.R' 'spatstat.R' 'sp.R' 'stars.R' 'subset.R' 'tidyverse.R' 'tile.R' 'transform.R' 'values.R' 'warp.R' 'write.R' 'xts.R'

NeedsCompilation no

Author Edzer Pebesma [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8049-7069>),
Michael Sumner [ctb] (ORCID: <https://orcid.org/0000-0002-2471-7511>),
Etienne Racine [ctb],
Adriano Fantini [ctb],
David Blodgett [ctb],
Krzysztof Dyba [ctb] (ORCID: <https://orcid.org/0000-0002-8614-3816>)

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

Repository CRAN

Date/Publication 2025-12-14 07:40:02 UTC

Contents

aggregate.stars . . . 3
as . . . 5
bcsd_obs . . . 6
c.stars . . . 6
contour.stars . . . 8
cut_stars . . . 8
dplyr . . . 9
expand_dimensions . . . 11
geom_stars . . . 11
L7_ETMs . . . 12
make_intervals . . . 13
mdim . . . 13
merge . . . 15
ops_stars . . . 16
plot . . . 17
prcomp . . . 21
predict.stars . . . 22
print_stars . . . 23
read_ncdf . . . 24
read_stars . . . 26
redimension . . . 28
stars_sentinel2 . . . 29
stars_subset . . . 29
st_apply . . . 32
st_as_sf . . . 33
st_as_stars . . . 35
st_cells . . . 39
st_contour . . . 40
st_coordinates . . . 41
st_crop . . . 42
st_dimensions . . . 44
st_dim_to_attr . . . 46

st_downsample 47
st_extract 48
st_geotransform 50
st_intersects.stars 51
st_join.stars 52
st_mosaic 53
st_rasterize 55
st_raster_type 56
st_res 57
st_rgb 58
st_rotate 59
st_set_bbox 60
st_tile 61
st_transform 62
st_warp 63
st_xy2sfc 64
write_stars 65
%in%,stars-method 67

Index **68**

`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 CFTime) 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), or a function that cuts time into intervals; 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. Note: each pixel is assigned to only a single group (in the order the groups occur) so non-overlapping spatial features and temporal windows are recommended.
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

See Also

aggregate, st_interpolate_aw, st_extract, https://github.com/r-spatial/stars/issues/317

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")
#TBD:
#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")
#TBD:
# x_agg_time - x_agg_posix
aggregate(x, "2 days", mean)
if (require(ncmeta, quietly = TRUE)) {
 # 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)
}
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

Coerce stars object into a terra SpatRaster

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.

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 SpatRaster.

Value

RasterLayer or RasterBrick

SpatRaster

`bcsd_obs` *Monthly Gridded Meteorological Observations*

Description

These are the monthly observational data used for BCSD downscaling. See: <https://gdo-dcp.ucllnl.org/downscaled_cmip_pr for more information." ; "Atmospheric Temperature, Air Temperature Atmosphere, Precipitation, Rain, Maximum Daily Temperature, Minimum Daily Temperature" ;

Usage

```
bcsd_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_proxy'
c(
  ...,
  along = NA_integer_,
  along_crs = FALSE,
  try_hard = FALSE,
  nms = names(list(...)),
  tolerance = sqrt(.Machine$double.eps)
```

```
)

## S3 method for class 'stars'
c(
  ...,
  along = NA_integer_,
  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
`along_crs`	logical; if `TRUE`, combine arrays along a CRS dimension
`try_hard`	logical; if `TRUE` and some arrays have different dimensions, combine those that dimensions matching to the first array
`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

Details

An error is raised when attempting to combine arrays with different measurement units into a single array. If this was intentded, `drop_units` can be used to remove units of a `stars` object before merging.

Value

a single `stars` object with merged (binded) arrays.

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_stars *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 *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, ...)

rename.stars(.data, ...)

rename.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, along, index, ...)

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)
if (require(dplyr, quietly = TRUE)) {
 x1 %>% slice("band", 2:3)
 x1 %>% slice("x", 50:100)
}
```

expand_dimensions *expand the dimension values into a list*

Description

expand the dimension values into a list

Usage

```
expand_dimensions(x, ...)

## S3 method for class 'dimensions'
expand_dimensions(x, ..., max = FALSE, center = NA)
```

Arguments

x	object of class ‘stars‘ or ‘dimensions‘
...	ignored
max	logical; if ‘TRUE‘ return the max (end) values of the dimensions intervals
center	logical; if ‘TRUE‘ return the center values of intervals, otherwise return offset (start) of intervals; if ‘NA‘ (default) return centers for x/y dimensions, offsets for all others

geom_stars *ggplot geom for stars objects*

Description

ggplot geom for stars objects

Usage

```
geom_stars(
  mapping = NULL,
  data = NULL,
  ...,
  downsample = 0,
  sf = FALSE,
  na.action = na.pass
)

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.
na.action	function; if NA values need to be removed before plotting use the value na.omit here (only applies to objects with raster dimensions)

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. If a stars array contains hex color values, and no fill parameter is given, the color values are used as fill color; see the example below.

If visual artefacts occur (Moiré-Effekt), then see the details section of plot.stars

Examples

```
system.file("tif/L7_ETMs.tif", package = "stars") %>% read_stars() -> x
if (require(ggplot2, quietly = TRUE)) {
  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))
  # plot rgb composite:
  st_as_stars(L7_ETMs)[,,,1:3] |> st_rgb() -> x # x contains colors as pixel values
  ggplot() + geom_stars(data = x)
}
```

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

`mdim` *Read or write data using GDAL's multidimensional array API*

Description

Read or write data using GDAL's multidimensional array API

Usage

```
read_mdim(
  filename,
  variable = character(0),
  ...,
  options = character(0),
  raster = NULL,
  offset = integer(0),
  count = integer(0),
  step = integer(0),
  proxy = FALSE,
  debug = FALSE,
  bounds = TRUE,
  curvilinear = NA,
  normalize_path = TRUE
)

write_mdim(
  x,
  filename,
  driver = detect.driver(filename),
  ...,
  root_group_options = character(0),
  options = character(0),
  as_float = TRUE,
  normalize_path = TRUE
)
```

Arguments

filename	name of the source or destination file or data source
variable	name of the array to be read; if '"?"', a list of array names is returned, with group name as list element names.
...	ignored
options	character; driver specific options regarding the opening (read_mdim) or creation (write_mdim) of the dataset
raster	names of the raster variables (default: first two dimensions)
offset	integer; zero-based offset for each dimension (pixels) of sub-array to read, defaults to 0 for each dimension(requires sf >= 1.0-9)
count	integer; size for each dimension (pixels) of sub-array to read (default: read all); a value of NA will read the corresponding dimension entirely; counts are relative to the step size (requires sf >= 1.0-9)
step	integer; step size for each dimension (pixels) of sub-array to read; defaults to 1 for each dimension (requires sf >= 1.0-9)
proxy	logical; return proxy object?
debug	logical; print debug info?

bounds	logical or character: if TRUE tries to infer from "bounds" attribute; if character, named vector of the form c(longitude="lon_bnds", latitude="lat_bnds") with names dimension names
curvilinear	control reading curvilinear (geolocation) coordinate arrays; if NA try reading the x/y dimension names; if character, defines the arrays to read; if FALSE do not try; see also read_stars
normalize_path	logical; if FALSE, suppress a call to normalizePath on filename
x	stars object
driver	character; driver name
root_group_options	character; driver specific options regarding the creation of the root group
as_float	logical; if TRUE write 4-byte floating point numbers, if FALSE write 8-byte doubles

Details

it is assumed that the first two dimensions are easting and northing

See Also

gdal_utils, in particular util mdiminfo to query properties of a file or data source containing arrays

Examples

```
set.seed(135)
m = matrix(runif(10), 2, 5)
names(dim(m)) = c("stations", "time")
times = as.Date("2022-05-01") + 1:5
pts = st_as_sfc(c("POINT(0 1)", "POINT(3 5)"))
s = st_as_stars(list(Precipitation = m)) |>
 st_set_dimensions(1, values = pts) |>
 st_set_dimensions(2, values = times)
nc = tempfile(fileext=".nc")
if (compareVersion(sf_extSoftVersion()["GDAL"], "3.4.0") > -1) {
  write_mdim(s, nc)
  # try ncdump on the generated file
  print(read_mdim(nc))
}
```

---

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 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

```
## 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 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

```
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 *plot stars object, with subplots for each level of first non-spatial dimension*

Description

plot stars object, with subplots for each level of first non-spatial dimension, and customization of legend key

Usage

```
## S3 method for class 'nc_proxy'
plot(x, y, ..., downsample = get_downsample(dim(x)), max_times = 16)

## S3 method for class 'stars'
plot(
  x,
```

```
  y,
  ...,
  join_zlim = TRUE,
  main = make_label(x, 1),
  axes = FALSE,
  downsample = TRUE,
  nbreaks = 11,
  breaks = "quantile",
  col = grey(1:(nbreaks - 1)/nbreaks),
  key.pos = get_key_pos(x, ...),
  key.width = kw_dflt(x, key.pos),
  key.length = 0.618,
  key.lab = ifelse(length(main) == 1, main, ""),
  reset = TRUE,
  box_col = NA,
  center_time = FALSE,
  hook = NULL,
  mfrow = NULL,
  compact = TRUE
)

## 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,
  text_values = FALSE,
  text_color = "black",
  axes = FALSE,
  interpolate = FALSE,
  as_points = FALSE,
  key.pos = NULL,
  logz = FALSE,
  key.width = kw_dflt(x, key.pos),
  key.length = 0.618,
  add.geom = NULL,
  border = NA,
  useRaster = isTRUE(dev.capabilities()$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.
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 number of pixels/lines/bands etc that will be skipped; see Details.
max_times	integer; maximum number of time steps to attempt to plot.
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?
nbreaks	number of color breaks; should be one more than number of colors. If missing and col is specified, it is derived from that.
breaks	numeric vector with actual color breaks, or a style name used in classIntervals.
col	colors to use for grid cells, or color palette function
key.pos	numeric; 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. If it has length 2, the second value, ranging from 0 to 1, determines where the key is placed in the available space (default: 0.5, center).
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)
key.lab	character; label for color key in case of multiple subplots, use "" to suppress
reset	logical; if FALSE, keep the plot in a mode that allows adding further map elements; if TRUE restore original mode after plotting
box_col	color for box around sub-plots; use NA 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; see examples.
mfrow	length-2 integer vector with nrows, ncolumns of a composite plot, to override the default layout

compact	logical; place facets compactly (TRUE), or spread over the plotting device area?
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 sfc, 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

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

Downsampling: a value for downsample of 0: no downsampling, 1: after every dimension value (pixel/line/band), one value is skipped (half of the original resolution), 2: after every dimension value, 2 values are skipped (one third of the original resolution), etc. If downsample is TRUE or a length 1 numeric vector, downsampling is only applied to the raster [x] and [y] dimensions.

To remove unused classes in a categorical raster, use the droplevels function.

When bitmaps show visual artefacts (Moiré effects), make sure that device png is used rather than ragg::agg_png as the latter uses antialiasing for filled polygons which causes this; see also https://github.com/r-spatial/stars/issues/573 .

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 has to be set manually.

Examples

```
st_bbox(L7_ETMs) |> st_as_sfc() |> st_centroid() |> st_coordinates() -> pt
hook1 = function() {
    text(pt[,"X"], pt[,"Y"], "foo", col = 'orange', cex = 2)
}
plot(L7_ETMs, hook = hook1)
x = st_set_dimensions(L7_ETMs, 3, paste0("B_", 1:6))
hook2 = function(..., row, col, nr, nrow, ncol, value, bbox) {
  str = paste0("row ", row, "/", nrow, ", col ", col, "/", ncol, "\nnr: ", nr, " value: ", value)
   bbox |> st_as_sfc() |> st_centroid() |> st_coordinates() -> pt
   text(pt[,"X"], pt[,"Y"], str, col = 'red', cex = 2)
}
plot(x, hook = hook2, col = grey(c(.2,.25,.3,.35)))
if (isTRUE(dev.capabilities()$rasterImage == "yes")) {
  lc = read_stars(system.file("tif/lc.tif", package = "stars"))
  levels(lc[[1]]) = abbreviate(levels(lc[[1]]), 6) # so it's not only legend
  plot(lc, key.pos=4)
}
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
image(x, col = grey((3:9)/10))
if (isTRUE(dev.capabilities()$rasterImage == "yes")) {
  image(x, rgb = c(1,3,5)) # false color composite
}
```

---

prcomp *Principle components of stars object*

Description

Compute principle components of stars object

Usage

```
## S3 method for class 'stars_proxy'
prcomp(x, ..., downsample = 0)

## S3 method for class 'stars'
prcomp(x, ..., quiet = FALSE)
```

Arguments

x	object of class ‘stars‘ or ‘stars_proxy‘
...	see prcomp
downsample	see st_as_stars
quiet	logical; if ‘TRUE‘, suppress message that PCs will be computed on last dimension; see details

Details

if ‘x‘ has only one attribute, principle components will be computed in the space of the last dimension of ‘x‘ to predict PC scores into a ‘stars‘ object, use predict.stars; see example below

Value

object of class ‘prcomp‘, see prcomp

Examples

```
l7 = split(st_as_stars(L7_ETMs), 3) # use bands as features
plot(prcomp(l7))
plot(merge(predict(l7, model = prcomp(l7))))
```

`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_proxy'
predict(object, model, ...)

## S3 method for class 'stars'
predict(object, model, ..., drop_dimensions = FALSE)
```

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
`drop_dimensions`	logical; if ‘TRUE‘, remove dimensions (coordinates etc) from ‘data.frame‘ with predictors

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)‘

print_stars *print stars or dimensions object*

Description

print stars or dimensions object

Usage

```
## S3 method for class 'dimensions'
as.data.frame(
  x,
  ...,
  digits = max(3, getOption("digits") - 3),
  usetz = TRUE,
  stars_crs = getOption("stars.crs") %||% 28,
  all = FALSE
)

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

## S3 method for class 'stars'
print(x, ..., n = 1e+05, abbrev = 30)
```

Arguments

x	object of class stars or of class dimensions
...	passed on to as.data.frame.dimensions
digits	number of digits to print numbers
usetz	logical; used to format POSIXct values
stars_crs	maximum width of string for CRS objects
all	logical; if TRUE print also fields entirely filled with NA or NULL
n	when prod(dim(x)) > 10 * n, the first n cells are used for attribute summary statistics
abbrev	number of characters to abbreviate attribute names to

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,
  ncsub = NULL,
  curvilinear = character(0),
  eps = sqrt(.Machine$double.eps),
  ignore_bounds = FALSE,
  make_time = TRUE,
  make_units = TRUE,
  proxy = NULL,
  downsample = 0
)
```

Arguments

.x	NetCDF file or source as a character vector or an nc_proxy object.
...	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 CFTime class for any "time" dimension.
make_units	if TRUE (the default), an attempt is made to set the units property of each variable
proxy	logical; if TRUE, an object of class stars_proxy is read which contains array metadata only; if FALSE the full array data is read in memory. If not set, defaults to TRUE when the number of cells to be read is larger than options(stars.n_proxy), or to 1e8 if that option was not set.

`downsample` integer; number of cells to omit between samples along each dimension. e.g. `c(1,1,2)` would return every other cell in x and y and every third cell in the third dimension (z or t). If 0, no downsampling is applied. Note that this transformation is applied AFTER NetCDF data are read using st_downsample. As such, if proxy=TRUE, this option is ignored.

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.

Any time dimension in the source is captured in a 'CFTime' object, unless the `make_time` argument is false. In that case, the numeric offsets representing time are returned instead.

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

`start` and `count` columns of ncsub 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

```
f <- system.file("nc/reduced.nc", package = "stars")
if (require(ncmeta, quietly = TRUE)) {
 read_ncdf(f)
 read_ncdf(f, var = c("anom"))
 read_ncdf(f, ncsub = cbind(start = c(1, 1, 1, 1), count = c(10, 12, 1, 1)))
}

if (require(ncmeta, quietly = TRUE)) {
 #' precipitation data in a curvilinear NetCDF
 prec_file = system.file("nc/test_stageiv_xyt.nc", package = "stars")
 prec = read_ncdf(prec_file, curvilinear = c("lon", "lat"), ignore_bounds = TRUE)
}

##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)))
}
if (require(dplyr, quietly = TRUE)) {
  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

```
read_stars(
  .x,
  sub = TRUE,
  ...,
  options = character(0),
  driver = character(0),
  quiet = FALSE,
  NA_value = NA_real_,
  along = NA_integer_,
  RasterIO = list(),
  proxy = getOption("stars.n_proxy") %||% 1e+08,
  curvilinear = character(0),
  normalize_path = TRUE,
  RAT = character(0),
  tolerance = 1e-10,
  exclude = "",
  shorten = TRUE
)
```

Arguments

.x	character vector with name(s) of file(s) or data source(s) to be read, or a function that returns such a vector
sub	character, integer or logical; name, index or indicator of sub-dataset(s) to be read
...	passed on to st_as_stars if curvilinear was set
options	character; opening options
driver	character; driver to use for opening file. To override fixing for subdatasets and autodetect them as well, use NULL.
quiet	logical; print progress output?
NA_value	numeric value to be used for conversion into NA values; by default this is read from the input file
along	length-one character or integer, or list; determines how several arrays are combined, see Details.

RasterIO	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.
proxy	logical; if TRUE, an object of class stars_proxy is read which contains array metadata only; if FALSE the full array data is read in memory. Always FALSE for curvilinear girds. If set to a number, defaults to TRUE when the number of cells to be read is larger than that number.
curvilinear	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
normalize_path	logical; if FALSE, suppress a call to normalizePath on .x
RAT	character; raster attribute table column name to use as factor levels
tolerance	numeric; passed on to all.equal for comparing dimension parameters.
exclude	character; vector with category value(s) to exclude
shorten	logical or character; if TRUE and length(.x) > 1, remove common start and end parts of array names; if character a new prefix

Details

In case .x contains multiple files, they will all be read and combined with c.stars. Along which dimension, or how should objects be merged? If along is set to 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 time indicates different time stamps. A single name (or positive value) for 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 along to specify them; see example.

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, resample. See https://gdal.org/en/latest/doxygen/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".

Data that are read into memory (proxy=FALSE) are read into a numeric (double) array, except for categorical variables which are read into an numeric (integer) array of class factor.

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))
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

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

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

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

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
name	character name of the new dimension

---

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)

Usage

```
stars_sentinel2
```

Format

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

---

stars_subset *subset stars objects*

Description

subset stars objects

Usage

```
## S3 replacement method for class 'stars_proxy'
x[i, downsample = 0] <- value

## 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, sfc, bbox, or stars used as spatial selector; see details
downsample	downsampling rate used in case i is a stars_proxy object
value	array of dimensions equal to those in x, or a vector or value that will be recycled to such an array
...	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.
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. If i is of class stars, and attributes of i are logical, cells in x corresponding to NA or FALSE cells in i are assigned an NA. Dimension ranges containing negative values or NA may be partially supported. Character selectors are matched against the names of a dimension if it has names, otherwise to the dimension values.

in an assignment (or replacement form, [<-), argument i needs to be either (i) a stars object with logical attribute(s) that has dimensions matching (possibly after recycling) those of x, in which case the TRUE cells will be replaced and i and/or value will be recycled to the dimensions of the arrays in x, or (ii) a length-one integer or character vector indicating which array to replace, in which case value may be stars object or a vector or array (that will be recycled).

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)
# with i of class stars:
x[x > 75] # generates lots of NA's; pattern for each band
x[x[,,,1] > 75] # recycles a single band template for all bands
x = read_stars(tif)
# replace, using a logical stars selector: cuts all values above 90 to 90
x[x > 90] = 90
# replace a single attribute when there are more than one:
s = split(x)
names(s) = paste0("band", 1:6)
# rescale only band 1:
s[1] = s[1] * 0.75
# rescale only attribute named "band2":
s["band2"] = s["band2"] * 0.85
# create a new attribute from a numeric vector:
s["rnorm"] = rnorm(prod(dim(s)))
s
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 *st_apply apply a function to one or more array dimensions*

Description

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

Usage

```
## 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),
  keep = FALSE
)
```

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).
keep	logical; if TRUE, preserve dimension metadata (e.g. time stamps)

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. Following the logic of apply, This new dimension is put before the other dimensions; use aperm to rearrange this, see last example.

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
## Not run:
 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)
   # note that we can select bands 3 and 4 in the first argument:
 ndvi2 = st_apply(x[,,,3:4], 1:2, fn_ndvi2)
 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_apply(x, 1:2, range) # dimension "range" is first; rearrange by:
 st_apply(x, 1:2, range) %>% aperm(c(2,3,1))

## End(Not run)
```

`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 *convert objects into a stars object*

Description

convert objects into a stars object

Usage

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

## S3 method for class 'CFDataset'
st_as_stars(
  .x,
  ...,
  .var = .x$var_names[1],
  all_compatible = length(.var) == 1
)

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

## S3 method for class 'cubble_df'
st_as_stars(.x, ..., check_times = FALSE)

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

## S3 method for class 'stars_proxy'
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, y_decreasing = TRUE, coords = 1:2)

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

## S3 method for class 'SpatRaster'
st_as_stars(
  .x,
  ...,
  ignore_file = FALSE,
  as_attributes = all(terra::is.factor(.x))
)

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

st_as_stars(.x, ...)

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

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

## S3 method for class 'stars'
st_as_stars(.x, ..., curvilinear = NULL, crs = st_crs("OGC:CRS84"))

## 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,
```

```
  proxy = FALSE
)

## S3 method for class 'xts'
st_as_stars(.x, ..., dimensions, name = "attr")
```

Arguments

.x	object to convert
...	in case .x is of class bbox, arguments passed on to pretty. In case .x is of class nc_proxy, arguments passed on to read_ncdf.
sf_geometry	sf data.frame with geometry and attributes to be added to stars object. Must have same number of rows as timeseries instances.
.var	Character vector with names of netCDF data variables to convert into stars attributes. All data variables have to use the same set of axes. If omitted, the first data variable will be used.
all_compatible	Logical flag that indicates if all data variables with the same axes as argument 'var' uses should be converted into attributes. Default is 'TRUE' when only one data variable is specified, 'FALSE' otherwise.
check_times	logical; should we check that the time stamps of all time series are identical?
as_col	logical; return rgb numbers (FALSE) or (character) color values (TRUE)?
downsample	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, see details
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
att	see factorValues; column in the RasterLayer's attribute table
ignore_file	logical; if TRUE, ignore the SpatRaster object file name
as_attributes	logical; if TRUE and .x has more than one layer, load these as separate attributes rather than as a band or time dimension (only implemented for the case where ignore_file is TRUE)
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 or stars arrays, or the names of the corresponding attributes in .x; the names of this vector should correspond to raster dimensions the matrices are associated with; see Details.

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 or object of class units; cell size in x direction; see details
dy	numeric or object of class units; 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 (TRUE), or always cover the bbox (FALSE), or find a good approximation (NA, default)?
nz	integer; number of cells in z direction; if missing no z-dimension is created.
proxy	logical; should a stars_proxy object be created? (requires gdal_create binary when sf < 1.0-6)
name	character; attribute name for array from an xts object

Details

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.

If xy is not specified and the first two dimensions in dims are both numeric, then it is set to these two dimensions.

The st_as_stars method for sf objects without any additional arguments returns a one-dimensional data cube with a dimension for the simple features geometries, and all remaining attributes as data cube attributes. When used with further arguments, the method for data.frames is called.

if curvilinear is a list with stars objects with longitude and latitude values, its coordinate reference system is typically not that of the latitude and longitude values. If curvilinear contains the names of two arrays in .x, then these are removed from the returned object.

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 and values is a matrix, the number of columns and rows of the matrix are taken.

Otherwise, if nx and ny are missing, they are computed as the (ceiling, floor, or rounded to integer value) of the ratio of the (x or y) range divided by (dx or dy), depending on the value of inside. Positive dy will be made negative. Further named arguments (...) are passed on to pretty. If dx or dy are units objects, their value is converted to the units of st_crs(.x) (only when sf >= 1.0-7).

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

Examples

```
if (require(plm, quietly = TRUE)) {
 data(Produc, package = "plm")
 st_as_stars(Produc)
}
if (require(dplyr, quietly = TRUE)) {
  # https://stackoverflow.com/questions/77368957/
spatial_dim <- st_sf(
  ID = 1:3,
  geometry = list(
    st_polygon(list(
      cbind(c(0, 1, 1, 0, 0), c(0, 0, 1, 1, 0))
    )),
    st_polygon(list(
      cbind(c(1, 2, 2, 1, 1), c(0, 0, 1, 1, 0))
    )),
    st_polygon(list(
      cbind(c(2, 3, 3, 2, 2), c(0, 0, 1, 1, 0))
    ))
  )
)
weekdays_dim <- data.frame(weekdays = c("Monday", "Tuesday", "Wednesday",
    "Thursday", "Friday", "Saturday", "Sunday"))
hours_dim <- data.frame(hours = c("8am", "11am", "4pm", "11pm"))
sf_dta <- spatial_dim |>
  cross_join(weekdays_dim)|>
  cross_join(hours_dim) |>
  mutate(population = rnorm(n(), mean = 1000, sd = 200)) |>
  select(everything(), geometry)

st_as_stars(sf_dta, dims = c("weekdays", "hours", "geometry"))
}
demo(nc, echo=FALSE,ask=FALSE)
st_as_stars(nc)
st_as_stars(st_drop_geometry(nc), dims = "NAME")
data.frame(expand.grid(x=1:5, y = 1:5), z = rnorm(25)) |> st_as_stars()
nc = st_read(system.file("gpkg/nc.gpkg", package="sf"))
st_as_stars(nc)
```

`st_cells` *return the cell index corresponding to the location of a set of points*

Description

If the object has been cropped without normalization, then the indices return are relative to the original uncropped extent. See `st_crop`

Usage

```
st_cells(x, sf)
```

Arguments

x	object of class stars
sf	object of class sf or sfc

Examples

```
set.seed(1345)
st_bbox(L7_ETMs) |>
  st_as_sfc() |>
  st_sample(10) -> pts
(x <- st_cells(L7_ETMs, pts))
# get the pixel values (first band only):
st_as_stars(L7_ETMs)[[1]][x]
# get pixel values for all bands:
st_as_stars(L7_ETMs) |> split() |> sapply(`[`, x)
# compare with st_extract():
st_as_stars(L7_ETMs) |> split() |> st_extract(pts)
```

---

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

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

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

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.
add_coordinates	logical; if 'TRUE', columns with dimension values preceed the array values, otherwise they are omitted
.x	object to be converted to a tibble

st_crop *crop a stars object*

Description

crop a stars object

Usage

```
## S3 method for class 'mdim'
st_crop(x, y, ...)

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

## 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),
  normalize = FALSE
)
```

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

normalize logical; if TRUE then pass the cropped object to st_normalize before returning. This typically changes the 'offset' field and resets the 'from' field to 1, and changes the bounding box of the returned object accordingly.

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]
# equivalent:
st_crop(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)

# 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],
ymax = offset[2] +  3.1 * 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)

# slightly larger bbox:
bb = st_bbox(c(xmin = offset[1] + 1.9 * res[1],
ymin = offset[2] + 11.1 * res[2],
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

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

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

st_dimensions(x) <- value

## S3 replacement method for class 'stars'
st_dimensions(x) <- value

## S3 replacement method for class 'stars_proxy'
st_dimensions(x) <- value

## S3 replacement method for class 'list'
st_dimensions(x) <- value

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

## Default S3 method:
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; setting special value NULL removes dimension values, for instance to remove curvilinear raster coordinates
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 explictly. 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

```
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(µm) -> 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"))
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

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

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_downsample *downsample stars or stars_proxy objects*

Description

downsample a stars or stars_proxy object either by skipping rows, columns and bands, or by computing a single value (e.g. the mean) from the sub-tiles involved

Usage

```
st_downsample(x, n, ...)

## S3 method for class 'stars'
st_downsample(x, n, ..., offset = 0, FUN)

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

Arguments

x	object of class stars or stars_proxy
n	integer; for each dimension the number of pixels/lines/bands etc that will be skipped; see Details.
...	arguments passed on to FUN (e.g., na.rm = TRUE to ignore missing values if FUN is mean)
offset	integer; offset(s) for downsampling, in pixels, starting at the offset of each dimension; should be smaller or equal to n
FUN	function; if given, downsampling will apply FUN to each of the the subtiles

Details

If all n == 0, no downsampling takes place; if it is 1, every second row/column/band is skipped, if it is 2, every second+third row/column/band are skipped, etc.

Downsampling a `stars_proxy` object returns a `stars` object, is equivalent to calling `st_as_stars(x, downsample = 2)`, and only downsamples the first two (x and y) dimensions.

Downsampled regular rasters keep their dimension offsets, have a cell size (delta) that is n[i]+1 times larger, and may result in a (slightly) different extent.

Note that terra's aggregate with `fact=2` corresponds to `st_downsample(x, n = 1, FUN = mean)`: `fact` is one larger than `n`.

Examples

```
(m = matrix(1:121, 11, 11))
(s = st_as_stars(m))
st_downsample(s, 1)
st_downsample(s, 1)[[1]]
st_downsample(s, 1, offset = 1)
st_downsample(s, 1, offset = 1)[[1]]
st_downsample(s, 1, offset = c(0,1))
st_downsample(s, 1, offset = c(0,1))[[1]]
st_downsample(s, 1, FUN = mean)
st_downsample(s, 1, FUN = mean)[[1]]
st_downsample(s, 1, offset = 1, FUN = mean)
st_downsample(s, 1, offset = c(0,1), FUN = mean)[[1]]
```

---

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,
  resampling = c("nearest", "bilinear", "cubic", "cubicspline"),
  sfc_attribute = NULL
)
```

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, or two-column matrix with coordinate points in rows, indicating where to extract values of x, or a stars object with geometry and temporal dimensions (vector data cube)
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 first temporal dimension (matching classes POSIXct, POSIXt, Date, or CFTime), 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
resampling	character; resampling method; for method cubic or cubicspline, 'stars_proxy' objects should be used and GDAL should have version >= 3.10.0
sfc_attribute	character; if at is of class stars should the aggregation be performed for the attribute geometry rather than the dimension geometry? If NULL (default), the aggregation is performed at the dimension geometries, else the name of the attribute geometry to perform the aggregation on. If the given attribute geometry does not exist, the aggregation defaults to the dimension geometry.

Details

points outside the raster are returned as NA values. For large sets of points for which extraction is needed, passing a matrix as to at may be much faster than passing an sf or sfc object.

Value

if at is of class matrix, a matrix with extracted values is returned; if at is of class stars and a temporal dimension was passed to time_column, a stars object with the original at dimensions and the extracted values as attributes. otherwise: 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, if this is not the case, 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, pnt) %>% st_as_sf()
st_extract(r[,,,1], pnt)
```

```
st_extract(r, st_coordinates(pnt)) # "at" is a matrix: return a matrix
# Extraction on non-POINT geometries
poly = st_buffer(pnt, 1000)
st_extract(r, poly)

# Extraction with time matching
rdate = c(r, r*2, along = "date")
dates = c(Sys.Date()-1, Sys.Date())
rdate = st_set_dimensions(rdate, "date", values = c(dates))

pntsf = st_sf(date = dates, geometry = pnt)
st_extract(split(rdate, "band"), pntsf) # POINT geometries

polysf = st_buffer(pntsf, 1000)
st_extract(split(rdate, "band"), polysf, time_column = "date") # POLYGON geometries

vdc = st_sf(rdm = rnorm(20), polygons = st_buffer(st_sample(st_bbox(pnt), 20), 500),
geometry = rep(pnt, 2), date = rep(dates, each = 10)) |>
st_as_stars(dims = c("geometry", "date"))

(vdc_new = st_extract(split(rdate, "band"), vdc)) # stars vector data cube
merge(vdc_new, name = "band")

### Extraction applied to the geometries inside the vector data cube (cell values)
(vdc_new2 = st_extract(split(rdate, "band"), vdc,
   sfc_attribute = "polygons")) # stars vector data cube
merge(vdc_new2, name = "band")
```

`st_geotransform` *get or set the geotransform, or rotation matrix*

Description

get or set the geotransform, or rotation matrix

Usage

```
st_geotransform(x, ...)

st_geotransform(x) <- value

## S3 replacement method for class 'stars'
st_geotransform(x) <- value
```

Arguments

x	object of class stars or dimensions
...	ignored
value	length 6 numeric vector, or 2 x 2 (scaled) rotation matrix

Examples

```
# using the "classical" rotation matrix, see https://en.wikipedia.org/wiki/Rotation_matrix :
rot = function(theta, dxdy = c(1., -1.)) {
   th = theta / 180 * pi
   matrix(c(cos(th), sin(th), -sin(th), cos(th)), 2, 2) %*%
   matrix(c(dxdy[2], 0, 0, dxdy[1]), 2, 2)
}
l = st_downsample(st_as_stars(L7_ETMs), 9) # save time in plotting
st_geotransform(l) = rot(20, c(28.5, 28.5)) # clockwise, 20 degrees, scale by cell size
plot(l[,,,1])
m = rot(20, c(1, 2))
g = expand.grid(x = 0:4, y = 0:4)
plot(g[1:2], asp = 1)
text(g[,1], g[,2], labels = seq_along(g[,1]), pos = 4)
g = t(m %*% t(as.matrix(g)))
points(g, col = 'red')
text(g[,1], g[,2], labels = seq_along(g[,1]), pos = 4, col = 'red')

m = matrix(1:20, 4)
s0 = st_as_stars(m)
s = s0
# dy > 0, clockwise rotation:
st_geotransform(s) = rot(10, c(1,1))
plot(s0, reset = FALSE)
plot(s, add = TRUE)
# dy < 0, counter clockwise rotation, + expansion in x-direction:
layout(1)
s0 = st_as_stars(st_bbox(s0), dx = 1)
s0$values = 1:20
s0
plot(s0, reset = FALSE)
s = s0
st_geotransform(s) = rot(10, c(2,1))
plot(s, add = TRUE)
```

---

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
sparse	logical; if TRUE, return the a sparse logical matrix (object of class ‘sgbp‘), if FALSE, return a logical matrix
...	ignored, or passed on to ‘st_intersects.sf‘ for curvilinear grids
as_points	logical, should grid cells be considered as points (TRUE) or polygons (FALSE)? Default: FALSE and warning emitted
transpose	logical; should the transpose of the ‘sgbp‘ object be returned?

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

---

st_join.stars *Spatially join a stars and an ‘sf‘ object*

Description

Spatially join a stars and an ‘sf‘ object

Usage

```
## 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

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

```
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

.x	object of class stars, or character vector with input dataset names
...	further input stars objects
dst	character; destination file name; this will be a VRT file with references to the source file(s), see details
options	character; options to the gdalbuildvrt command
file_ext	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; note that for the ‘stars‘ method, the ‘dst‘ file may contain references to temporary files that are going to be removed at termination of the R session.

uses gdal_utils to internally call `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 `gdalbuildvrt`

Examples

```
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))
```

st_rasterize *rasterize simple feature geometries*

Description

rasterize simple feature geometries

Usage

```
st_rasterize(
  sf,
  template = guess_raster(sf, ...) %||% st_as_stars(st_bbox(sf), values = NA_real_,
    ...),
  file = tempfile(),
  driver = "GTiff",
  options = character(0),
  align = FALSE,
  proxy = FALSE,
  ...
)
```

Arguments

sf	object of class sf
template	optional; stars object with desired target geometry, or target geometry alignment if align=TRUE; see details
file	temporary file name
driver	driver for temporary file
options	character; options vector for GDALRasterize
align	logical; if TRUE, template is only used for the geometry _alignment_, informing target resolution and offset
proxy	logical; should a proxy object be returned?
...	arguments passed on to st_as_stars

Details

if ‘template‘ is a ‘stars‘ object, non-NA cells that are not covered by ‘sf‘ receive the value in ‘template‘; see also argument ‘align‘.

Examples

```
demo(nc, echo = FALSE, ask = FALSE)
(x = st_rasterize(nc)) # default grid:
plot(x, axes = TRUE)
# a bit more customized grid:
(x = st_rasterize(nc, st_as_stars(st_bbox(nc), nx = 100, ny = 50, values = NA_real_)))
```

```
plot(x, axes = TRUE)
(ls = st_sf(a = 1:2, st_sfc(st_linestring(rbind(c(0.1, 0), c(1.1, 1))),
   st_linestring(rbind(c(0, 0.05), c(1, 0.05))))))
(grd = st_as_stars(st_bbox(ls), nx = 10, ny = 10, xlim = c(0, 1.0), ylim = c(0, 1),
   values = NA_real_))
# Only the left-top corner is part of the grid cell:
sf_extSoftVersion()["GDAL"]
plot(st_rasterize(ls, grd), axes = TRUE, reset = FALSE) # ALL_TOUCHED=FALSE;
plot(ls, add = TRUE, col = "red")
plot(st_rasterize(ls, grd, options = "ALL_TOUCHED=TRUE"), axes = TRUE, reset = FALSE)
plot(ls, add = TRUE, col = "red")
# add lines to existing 0 values, summing values in case of multiple lines:
(grd = st_as_stars(st_bbox(ls), nx = 10, ny = 10, xlim = c(0, 1.0), ylim = c(0, 1), values = 0))
r = st_rasterize(ls, grd, options = c("MERGE_ALG=ADD", "ALL_TOUCHED=TRUE"))
plot(r, axes = TRUE, reset = FALSE)
plot(ls, add = TRUE, col = "red")
```

`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

```
st_raster_type(x, dimension = character(0))
```

Arguments

`x`	object of class `stars`
`dimension`	optional: numbers or names of dimension(s) to get per-dimension type

Details

categories `"curvilinear"` and `"affine"` only refer to the relationship between a pair of spatial (raster) dimensions.

Value

if `dimension` is not specified, return the spatial raster type: one of `NA` (if the object does not have raster dimensions), `"curvilinear"`, `"rectilinear"`, `"affine"`, or `"regular"`. In case dimension(s) are specified, return one of `"regular"`, `"rectilinear"` (irregular but numeric), or `"discrete"` (anything else).

Examples

```
tif = system.file("tif/L7_ETMs.tif", package = "stars")
x = read_stars(tif)
st_raster_type(x)
st_raster_type(x, 1:3)
```

---

st_res *obtain (spatial) resolution of a stars object*

Description

obtain resolution(s) of a stars object: by default only the (absolute) x/y raster dimensions, optionally all `delta` dimension parameters

Usage

```
st_res(x, all = FALSE, absolute = !all)
```

Arguments

`x`	an object of class `stars`
`all`	logical; if FALSE return a vector with the x/y raster resolution
`absolute`	logical; only works when `all = FALSE`; if TRUE return absolute resolution values, if FALSE return `delta` values

Value

if `all = FALSE` a vector with x/y raster resolutions, otherwise a list with delta values

Examples

```
st_res(L7_ETMs)
st_res(L7_ETMs, absolute = FALSE)
st_res(L7_ETMs, all = TRUE)
if (require(starsdata)) {
  paste0("netcdf/", c("avhrr-only-v2.19810901.nc",
    "avhrr-only-v2.19810902.nc",
    "avhrr-only-v2.19810903.nc",
    "avhrr-only-v2.19810904.nc")) |>
  system.file(package = "starsdata") |>
  read_stars(quiet = TRUE) -> x
  st_res(x) |> print()
  st_res(x, all = TRUE) |> print()
}
```

st_rgb *reduce dimension to rgb (alpha) hex values*

Description

reduce dimension to rgb (alpha) hex values

Usage

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

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 by "percent".
stretch	logical or character; if TRUE or "percent", each band is stretched to 0 ... maxColorValue by "percent clip" method using probs values. If "histogram", a "histogram equalization" is performed (probs values are ignored). If stretch is NULL or FALSE, no stretching is performed. Other character values are interpreted as "percent" and a message will be printed.

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. Alternatively, you can use plot.stars with the rgb argument to create a three-band composition.

See Also

st_apply, rgb

Examples

```
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()
}
r = st_rgb(x[,,,3:1],
   probs = c(0.01, 0.99),
   stretch = "percent")
plot(r)
r = st_rgb(x[,,,3:1],
   probs = c(0.01, 0.99),
   stretch = "histogram")
plot(r)
```

---

st_rotate *Transform rotated pole long/lat regular grid to unrotated curvilinear grid*

Description

Transform rotated long/lat regular grid to unrotated curvilinear grid

Usage

```
## S3 method for class 'stars'
st_rotate(.x, lon0, lat0, north = TRUE, ...)

## S3 method for class 'sfc'
st_rotate(.x, lon0, lat0, north = TRUE, ...)

## S3 method for class 'sf'
st_rotate(.x, lon0, lat0, north = TRUE, ...)
```

Arguments

.x	object of class stars
lon0	longitude of the rotated pole in degrees
lat0	latitude of the rotated pole in degrees
north	logical; if TRUE the pole refers to the North pole, otherwise the South pole
...	ignored

Value

curvilinear stars object with coordinates in regular long/lat (North pole at lat=90)

Examples

```
if (require("starsdata") && require("maps")) {
  # data downloaded from https://esgf-data.dkrz.de/search/cosmo-rea/
  nc = "netcdf/ts_EUR-6km_ECMWF-ERAINT_REA6_r1i1p1f1_COSMO_v1_mon_201801-201812.nc"
  f = system.file(nc, package = "starsdata")
  m = read_mdim(f, "ts")
  print(m)
  # NOTE this function is obsolete when reading m as
  # m = read_mdim(f, "ts", curvilinear = c("longitude", "latitude"))
  if (require(RNetCDF)) {
     x = open.nc(f)
     lon = att.get.nc(x, "rotated_latitude_longitude", "grid_north_pole_longitude")
     lat = att.get.nc(x, "rotated_latitude_longitude", "grid_north_pole_latitude")
     close.nc(x)
     print(c(lon = lon, lat = lat))
  } else {
     lon = -162
     lat = 39.25
  }
  m1 = st_rotate(m, lon, lat)
  print(m1)
  h = function() maps::map(add = TRUE)
  plot(m1, downsample = c(10, 10, 5), axes = TRUE, hook = h, mfrow = c(1, 2))
    # curvilinear grid: downsample for plotting speed
  m2 = st_warp(m1, crs = st_crs("OGC:CRS84"), threshold = .1)
  plot(m2, hook = h, mfrow = c(3, 4)) # regular grid: plots fast
}
```

`st_set_bbox` *set bounding box parameters of regular grid*

Description

set bounding box parameters of regular grid

Usage

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

Arguments

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

st_tile *Specify parameters to load raster in blocks*

Description

Helper function for specifying the block parameters (nXOff, nYOff, nXsize, and nYSize) required by RasterIO argument in read_stars

Usage

```
st_tile(img_rows, img_cols, x_window, y_window, overlap = 0)
```

Arguments

img_rows	number of input raster rows (integer)
img_cols	number of input raster columns (integer)
x_window	number of rows in block (integer)
y_window	number of columns in block (integer)
overlap	number of overlapping pixels (integer)

Value

matrix with specified nXOff, nYOff, nXsize, and nYSize parameters for every block

Examples

```
## Not run:
tif = system.file("tif/L7_ETMs.tif", package = "stars")
r = read_stars(tif, proxy = TRUE)
tiles = st_tile(nrow(r), ncol(r), 256, 256)
for (i in seq_len(nrow(tiles))) {
  tile = read_stars(tif, proxy = FALSE, RasterIO = tiles[i, ])
  # write tiles to separate files
  write_stars(tile, dsn = paste0(i, ".tif"))
}

## End(Not run)
```

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, ...)

st_transform_proj.stars(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).

If array values contain geometries and an array as a whole is of class ‘sfc‘ and has a non-missing CRS, array geometries are also transformed.

See Also

st_warp

Examples

```
geomatrix = system.file("tif/geomatrix.tif", package = "stars")
(x = read_stars(geomatrix))
new = st_crs('OGC:CRS84')
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 an new coordinate reference system*

Description

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

Usage

```
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",
  threshold = NA_real_
)
```

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 gdal's warp or warper, 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; not setting this when use_gdal is TRUE leads to a warning
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
threshold	numeric; distance threshold for warping curvilinear grids: new cells at distances larger than threshold are assigned NA values.

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. 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

```
geomatrix = system.file("tif/geomatrix.tif", package = "stars")
(x = read_stars(geomatrix))
new_crs = st_crs('OGC:CRS84')
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('OGC:CRS84') %>% 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) and vice versa*

Description

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

Usage

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

st_sfc2xy(x, ...)
```

Arguments

x	object of class stars, or of class sf
as_points	logical; if TRUE, generate points at cell centers, else generate polygons
...	for 'st_xy2sfc': arguments passed on to st_as_sfc, for 'st_sfc2xy' arguments passed on to st_as_stars.data.frame
na.rm	logical; omit (remove) cells which are entirely missing valued (across other dimensions)?

Details

'st_xy2sfc' replaces x y raster dimensions with simple feature geometry list (points, or polygons = rasterize)

'st_sfc2xy' replaces POINT simple feature geometry list with an x y raster

Value

'st_xy2sfc' returns an 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.

'st_sfc2xy' returns an 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.

Examples

```
(reduced_nc = read_ncdf(system.file("nc/reduced.nc", package = "stars")))
(x = stars::st_xy2sfc(reduced_nc, as_points = TRUE, na.rm = FALSE))
# roundtrip:
st_sfc2xy(x, dims=c("X", "Y","zlev","time"))
```

---

write_stars *write stars object to gdal dataset (typically: to file)*

Description

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

Usage

```
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,
  scale_offset = c(1, 0)
)

## S3 method for class 'stars_proxy'
write_stars(
  obj,
  dsn,
  layer = 1,
  ...,
  driver = detect.driver(dsn),
  options = character(0),
  scale_offset = c(1, 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
scale_offset	length 2 numeric vector with scale, offset values: raw values computed by raw = (value - offset) / scale are written to dsn; scale and offset values are written to dsn or else a warning is raised

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, 6
    L7_ETMs, 12
    stars_sentinel2, 29
[.stars, *58*
[.stars *(*stars_subset*)*, 29
[<-.stars *(*stars_subset*)*, 29
[<-.stars_proxy *(*stars_subset*)*, 29
%in%,stars-method, 67

aes, *12*
aggregate, *4*, *48*
aggregate *(*aggregate.stars*)*, 3
aggregate.stars, 3, *49*
all.equal, *7*, *27*
aperm, *33*
apply, *32*, *33*
as, 5
as.data.frame.dimensions *(*print_stars*)*, 23
as.data.frame.stars *(*st_coordinates*)*, 41
as.tbl_cube.stars *(*dplyr*)*, 9
as_tibble.stars *(*st_coordinates*)*, 41

bcsd_obs, 6

c.stars, 6, *27*
c.stars_proxy *(*c.stars*)*, 6
classIntervals, *19*
coerce,stars,Raster-method *(*as*)*, 5
coerce,stars,Terra-method *(*as*)*, 5
coerce,stars_proxy,Raster-method *(*as*)*, 5
coerce,stars_proxy,Terra-method *(*as*)*, 5
contour, *8*, *41*
contour.stars, 8
coord_equal, *12*
coverage_fraction, *4*
cut, *9*
cut.array *(*cut_stars*)*, 8
cut.matrix *(*cut_stars*)*, 8
cut.POSIXt, *4*
cut.stars *(*cut_stars*)*, 8
cut_stars, 8

detect.driver *(*write_stars*)*, 65
dplyr, 9
droplevels, *20*

expand_dimensions, 11

facet_wrap, *12*
factorValues, *37*
filter, *10*
filter.stars *(*dplyr*)*, 9
filter.stars_proxy *(*dplyr*)*, 9
findInterval, *4*

gdal_utils, *15*, *54*, *63*
gdal_write, *66*
geom_raster, *12*
geom_sf, *12*
geom_stars, 11
geom_tile, *12*

image.stars *(*plot*)*, 17

L7_ETMs, 12

make_intervals, 13
makeCluster, *32*
Math.stars *(*ops_stars*)*, 16
Math.stars_proxy *(*ops_stars*)*, 16
mdim, 13
merge, 15
mutate.stars *(*dplyr*)*, 9
mutate.stars_proxy *(*dplyr*)*, 9

normalizePath, *15*, *27*

Ops.stars *(*ops_stars*)*, 16
Ops.stars_proxy *(*ops_stars*)*, 16

ops_stars, 16

plot, 17
plot.stars, *12*, *58*
png, *20*
prcomp, *21*, 21, *22*
predict.stars, *22*, 22
predict.stars_proxy *(*predict.stars*)*, 22
pretty, *37*, *38*
print.dimensions *(*print_stars*)*, 23
print.stars *(*print_stars*)*, 23
print_stars, 23
projectRaster, *64*
pull, *10*
pull.stars *(*dplyr*)*, 9
pull.stars_proxy *(*dplyr*)*, 9

rasterImage, *20*
read_mdim *(*mdim*)*, 13
read_ncdf, 24, *37*
read_stars, *7*, *15*, 26, *61*, *66*
redimension, 28
rename.stars *(*dplyr*)*, 9
rename.stars_proxy *(*dplyr*)*, 9
replace_na, *10*
replace_na.stars *(*dplyr*)*, 9
replace_na.stars_proxy *(*dplyr*)*, 9
rgb, *20*, *58*

select.stars *(*dplyr*)*, 9
select.stars_proxy *(*dplyr*)*, 9
slice.stars *(*dplyr*)*, 9
slice.stars_proxy *(*dplyr*)*, 9
split *(*merge*)*, 15
st_apply, 32, *58*
st_as_sf, *4*, *20*, 33, *37*, *41*, *53*
st_as_sfc.stars *(*st_as_sf*)*, 33
st_as_stars, *21*, *26*, *34*, 35, *55*
st_as_stars.data.frame, *65*
st_cells, 39
st_contour, *8*, 40
st_coordinates, 41
st_crop, *30*, *39*, 42
st_dim_to_attr, 46
st_dimensions, 44
st_dimensions<- *(*st_dimensions*)*, 44
st_downsample, 47
st_drivers, *66*
st_extract, *4*, 48
st_flip *(*stars_subset*)*, 29
st_geotransform, 50
st_geotransform<- *(*st_geotransform*)*, 50
st_get_dimension_values
*(*st_dimensions*)*, 44
st_interpolate_aw, *4*
st_intersects.stars, 51, *53*
st_join.stars, 52
st_make_valid, *34*
st_mosaic, 53
st_normalize, *43*
st_raster_type, 56
st_rasterize, 55
st_redimension *(*redimension*)*, 28
st_res, 57
st_rgb, 58
st_rotate, 59
st_set_bbox, 60
st_set_dimensions *(*st_dimensions*)*, 44
st_sfc2xy *(*st_xy2sfc*)*, 64
st_tile, 61
st_transform, *62*, 62
st_transform_proj.stars *(*st_transform*)*,
62
st_warp, *62*, 63
st_xy2sfc, 64
stars_sentinel2, 29
stars_subset, 29

theme_stars *(*geom_stars*)*, 11
transmute.stars *(*dplyr*)*, 9
transmute.stars_proxy *(*dplyr*)*, 9

var.get.nc, *25*

write_mdim *(*mdim*)*, 13
write_stars, 65