

Package ‘ursa’

December 19, 2025

Type Package

Title Non-Interactive Spatial Tools for Raster Processing and Visualization

Version 3.11.5

Author Nikita Platonov [aut, cre] (ORCID:
<<https://orcid.org/0000-0001-7196-7882>>)

Maintainer Nikita Platonov <platonov@sev-in.ru>

Description S3 classes and methods for manipulation with georeferenced raster data: reading/writing, processing, multi-panel visualization.

License GPL (>= 2)

URL <https://github.com/nplatonov/ursa>

BugReports <https://github.com/nplatonov/ursa/issues>

Depends R (>= 4.1.0)

Imports utils, graphics, grDevices, stats, sf (>= 0.6-1)

Suggests proj4, raster, ncdf4, locfit, knitr, rmarkdown, tcltk, sp, methods, fasterize, IRdisplay, caTools, shiny, tools, png, jpeg, webp, htmlwidgets, htmltools, leaflet, leafem, leafpop, RColorBrewer, ragg, widgetframe, yyjsonr, geojsonsf (>= 2.0.0), leaflet.providers, magick, terra, stars, vapour, gdalraster, sys, RSQLite, whitebox

NeedsCompilation yes

ByteCompile no

Repository CRAN

Date/Publication 2025-12-19 18:40:02 UTC

Contents

ursa-package	4
allocate	4
as.array	6

as.data.frame	7
as.integer	9
as.matrix	10
as.Raster	12
as.raster	14
as.table	15
as.ursa	16
as_stars	19
bandname	20
band_group	21
band_stat	22
blank	24
c	25
chunk	26
close	28
codec	29
colorize	30
colortable	33
commonGeneric	36
compose_close	37
compose_design	39
compose_legend	43
compose_open	45
compose_panel	49
compose_plot	50
create_envi	51
cubehelix	54
dim	56
discolor	57
display	58
display_brick	60
display_rgb	61
display_stack	62
envi_files	64
Extract	65
focal_extrem	67
focal_mean	69
focal_median	70
focal_special	72
get_earthdata	74
glance	76
global operator	80
groupGeneric	82
head	84
hist	85
identify	86
ignorevalue	89
is.na	90

legend_align	91
legend_colorbar	93
legend_mtext	96
local_group	98
local_stat	100
na.omit	101
nband	102
open_envi	103
open_gdal	104
panel_annotation	106
panel_coastline	108
panel_contour	112
panel_decor	115
panel_graticule	116
panel_new	119
panel_plot	122
panel_raster	124
panel_scalebar	126
panel_shading	129
pixelsize	130
plot	132
polygonize	133
read_envi	135
read_gdal	137
reclass	138
regrid	140
rep	144
Replace	145
seq	147
session	149
sort	151
spatial_engine	152
spatial_levelsplit	157
spatial_read	158
spatial_write	159
summary	161
temporal_interpolate	162
temporal_mean	163
trackline	165
ursa	166
ursaConnection	168
ursaCRS	171
ursaGrid	172
ursaProgressBar	173
ursaRaster	175
ursaStack	177
ursaValue	179
ursa_cache	181

ursa_crop 182

ursa_crs 183

ursa_dummy 184

ursa_grid 186

ursa_info 187

ursa_match 189

ursa_new 190

whiteboxing 192

write_envi 194

write_gdal 195

zonal_stat 197

Index 199

ursa-package	Overview
--------------	----------

Description

Have a great work with **ursa**!

Details

See table of content.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

allocate	Rasterization of point data into grid cells
----------	---

Description

allocate takes x and y coordinates and values from data frame, which is describing point spatial data, and puts them into cells of raster. The certain function (either mean value, sum of values, number of points) is applied for >0 points inside of the exact cell borders.

Usage

```
allocate(vec, coords = c("x", "y"), nodata = NA, attr = "+",
        fun = c("mean", "sum", "sd", "n"), cellsize = NA, resetGrid = FALSE,
        verbose = FALSE)
```

Arguments

vec	data.frame. At least x and y should be in colnames(vec). It is allowed to use "SpatialPointsDataFrame" from package sp . The "on the fly" reprojection is not supported.
coords	Character of length 2. Columns names, which contain coordinates of data points. Raster bands are not produced for specified columns. For misreference of coordinate columns, the attempt to find more appropriate coordinate columns is taken.
fun	Character keyword of function, which is applied to value of points, which are dropped into the same cell. Valid values are "mean" (mean value), "sum" (sum of values), "sd" (standard deviation), "n" (number of points).
nodata	Numeric of length 1. This value used to mark NA values in the writing to file.
attr	Pattern in the format of regular expressions , which is used to select required columns in data frame. By default (".*") all columns are used.
cellsize	Numeric. Desired size of cell in the raster grid. Used only when source data are not in regular grid. Default is NA; cell size is determined automatically to exclude case of points overlapping.
resetGrid	Logical. If TRUE then existing base grid (from session_grid()) will be overwritten. Otherwise using of current grid will be attempted.
verbose	Logical. Some output in console. Primarily for debug purposes.

Details

Here fun differs from R-styled fun in such functions as [*apply](#), [aggregate](#).

It was refused "rasterize" for function name to distinguish with [rasterize](#) in the package **raster**

Value

Object of class `ursaRaster`

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
g1 <- session_grid(regrid(session_grid()),mul=1/10))
n <- 1000
x <- with(g1,runif(n,min=minx,max=maxx))
y <- with(g1,runif(n,min=miny,max=maxy))
z <- with(g1,runif(n,min=0,max=10))
da <- data.frame(x=x,y=y,value=z)
res <- c(mean=allocate(da,fun="mean")
        ,mean_=NA
        ,sum=allocate(da,fun="sum")
        ,count=allocate(da,fun="n"))
```

```
res["mean_"]=res["sum"]/res["count"]
print(res)
```

as.array

Export raster object to multidimensional array

Description

In the `ursaRaster` object the 3-dimensional image data are presented in 2-dimensional matrix. `as.array` transforms internal 2-dimensional data to the usual 3-dimensional data. `as.matrix` just extracts image data in internal 2-dimensional format.

Usage

```
## S3 method for class 'ursaRaster'
as.array(x, ...)

## non-public
.as.array(x, drop = FALSE, flip = FALSE, permute = FALSE, dim = FALSE)
```

Arguments

<code>...</code>	Arguments, which are passed to <code>.as.array</code> .
<code>x</code>	<code>ursaRaster</code> object
<code>drop</code>	Logical. If <code>drop=TRUE</code> then single-band images are presented without third dimension.
<code>permute</code>	Logical. If <code>permute=FALSE</code> then returned array has dimension (<i>samples, lines, bands</i>). If <code>permute=TRUE</code> then returned array has dimension (<i>lines, samples, bands</i>).
<code>flip</code>	Logical. If <code>flip=TRUE</code> then vertical flip (reverse coordinates for dimension #2) is applied for output image.
<code>dim</code>	Logical. If <code>dim=TRUE</code> then array's dimension is returned.

Details

Use `permute=TRUE` to create an object of class `raster`: `as.raster(as.array(...))`

The spatial reference system is lost.

Value

If `dim=FALSE` then `as.array` returns object of class `array`.

If `dim=TRUE` then `as.array` returns dimension of array.

`as.matrix` returns object of class `matrix`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[as.raster](#) is a function to direct export to the object of class [raster](#).
[as.matrix](#) with argument/value coords=TRUE and [as.data.frame](#) for object of class `ursaRaster` keep spatial reference system.

Examples

```
session_grid(NULL)
a <- pixelsize()
a <- (a-global_min(a))/(global_max(a)-global_min(a))
b <- c(entire=a, half=a/2, double=a*2)
str(m <- as.matrix(b))
str(d1 <- as.array(b))
str(d2 <- as.array(b[1], drop=FALSE))
str(d3 <- as.array(b[1], drop=TRUE))
contour(d3)
filled.contour(d3)
d4 <- as.array(b, perm=TRUE)/global_max(b)
d4[is.na(d4)] <- 0
str(d4 <- as.raster(d4))
plot(d4)
```

as.data.frame	<i>Convert raster image to a data frame</i>
---------------	---

Description

`as.data.frame` reorganizes `ursaRaster` object into data frame, where first two columns (x and y) are coordinates of cells, and the rest columns are cell values.

Usage

```
## S3 method for class 'ursaRaster'
as.data.frame(x, ...)

# non-public
.as.data.frame(obj, band = FALSE, id = FALSE, na.rm = TRUE, all.na = FALSE,
               col.names = NULL)
```

Arguments

- `x, obj` Object of class `ursaRaster`
- `...` Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes. Passed to non-public `.as.data.frame`.

Pattern	as.data.frame)	Argument	(.as.data.frame)	Description
---------	----------------	----------	------------------	-------------

band	band	<i>See below.</i>
id	id	<i>See below.</i>
na\\.rm	na.rm	<i>See below.</i>
all\\.na	all.na	<i>See below.</i>
col(\\.)*name(s)*	col.names	<i>See below.</i>

band	Logical. If band=FALSE then each band is presented by separate column in the data frame. If band=TRUE then band name is presented as a factor in the column \$band, and values are written in the column \$z. If band=TRUE then number of rows is
id	Logical. If band=FALSE then is ignored. If id=TRUE then additional columns \$id will contain unique cell number in the source raster.
na.rm	Logical. If na.rm=FALSE then number of rows for data frame is equal to number of cells of spatial grid of raster. If na.rm=TRUE then cells with 'no data' values for all (all.na=FALSE) or any (all.na=TRUE) bands are omitted.
all.na	Logical. If na.rm=FALSE then ignored. If number of rows for data frame is equal to number of cells of spatial grid of raster. If na.rm=TRUE then cells with 'no data' values for all bands are omitted.
col.names	Character vector or NULL. Names for columns of data frame. If NULL, then column names are generated from band names. Default is NULL.

Details

The structure of voxel is kept. The number of rows for band=TRUE is equal to the number of rows for band=FALSE multiplied to number of bands. To extract all numeric data with destroying of voxel, you may use followed code:

```
subset(as.data.frame(obj,band=TRUE),!is.na(z)).
```

Value

Data frame.

If band=TRUE then

x	Horizontal coordinate of cell's midpoint
y	Vertical coordinate of cell's midpoint
z	Value
band	Band as a factor
id	<i>Optional.</i> Unique number for (x,y) coordinate.

If band=FALSE then

x	Horizontal coordinate of cell's midpoint
y	Vertical coordinate of cell's midpoint
...	Additional columns. Names of columns are names of bands. Values of columns are values of corresponded bands.

If ursaRaster is projected, then data frame has additional attribute `attr(...,"proj")` with value of PROJ.4 string.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
session_grid(regrid(res=50000,lim=c(-1200100,-1400800,1600900,1800200)))
a0 <- ursa_dummy(nband=3,min=0,max=100)
a0[a0<30 | a0>70] <- NA
names(a0) <- c("x","y","z")
print(a0)
b0 <- as.data.frame(a0)
session_grid(NULL)
a1 <- as.ursa(b0)
print(a1-a0)
session_grid(NULL)
session_grid(regrid(res=5800000))
set.seed(352)
a2 <- as.integer(ursa_dummy(nband=2,min=0,max=100))
a2[a2>50] <- NA
print(a2)
print(b1 <- as.data.frame(a2,na.rm=FALSE))
print(b2 <- as.data.frame(a2,na.rm=TRUE))
print(b3 <- as.data.frame(a2,all.na=TRUE))
print(b4 <- as.data.frame(a2,band=TRUE,na.rm=FALSE))
print(b5 <- as.data.frame(a2,band=TRUE,all.na=FALSE))
print(b6 <- as.data.frame(a2,band=TRUE,all.na=TRUE))
print(b7 <- as.data.frame(a2,band=TRUE,all.na=TRUE,id=TRUE))
```

as.integer

Transform values to type integer

Description

as.integer for object of class ursaRaster truncates decimal part of image values and then converts to type [integer](#).

Usage

```
## S3 method for class 'ursaRaster'
as.integer(x, ...)
```

Arguments

x	ursaRaster object
...	Other arguments which passed to function as.integer of package base .

Value

Object of class `ursaRaster` where `storage.mode` of values is integer.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- pixelsize()
a <- a-min(a)+0.5
str(ursa_value(a))
print(storage.mode(a$value))
b <- as.integer(a)
str(ursa_value(b))
print(storage.mode(b$value))
```

as.matrix

Convert raster image to a matrix

Description

`as.matrix(coords=TRUE)` prepares a list from the first band of `ursaRaster`, which is suitable as input parameter for functions `image`, `contour` and `filled.contour`.

Usage

```
## S3 method for class 'ursaRaster'
as.matrix(x, ...)
## S3 method for class 'ursaRaster'
x[[i]]
```

Arguments

x	Object of class <code>ursaRaster</code>
...	Set of arguments, which are recognized via their names (using regular expressions) and classes. (<code>coord(s)* crd ^\$</code>) Logical If TRUE then list is created with x, y, z components, where component \$z contains matrix, components \$x and \$y are coordinates for elements if matrix.
i	Positive integer or character of length. If integer, then band index. If character, then band name. If missing, then first band (value 1L) is used.

Details

Item colortable is mainly for internal usage, e. g., for mapping. Item proj is useful for conversion back to ursaRaster object by calling [as.ursa](#) function.

Extract operator `x[[i]]` is a wrapper for `as.matrix(x[i], coords=TRUE)`

Value

Depending of argument `coords`.

If `coords=FALSE`, then it is a two-dimensional matrix `c(samples*lines, bands)`, [unclassed](#) from `ursaValue` class.

If `coords=TRUE`, then it is a list:

<code>x</code>	Numeric. Midpoints of cells on horizontal axis
<code>y</code>	Numeric. Midpoints of cells on vertical axis
<code>z</code>	Numeric. Matrix of values
<code>attr(*, "proj")</code>	PROJ.4 string for grid, defined by <code>x</code> and <code>y</code>
<code>attr(*, "colortable")</code>	Optional. Object of class ursaColorTable . Missing if raster has no color table.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- ursa_dummy(nband=3,min=0,max=100)
a <- a[a>=20 & a<=80]
ignorevalue(a) <- 121
str(ursa_value(a[2]))
str(as.matrix(a[2]))
b1 <- a[[2]]
str(b1)
image(b1,asp=1)
b2 <- as.matrix(a[2:3],coords=TRUE)
print(c('theSame?'=identical(b1,b2)))
a2 <- as.ursa(b2)
res <- c(src=a[2],exported_then_imported=a2,diff=a[2]-a2)
print(res)
```

as.Raster

*Coercion to package 'raster' objects***Description**

as.Raster converts single-band `ursaRaster` object to *raster*, multi-band `ursaRaster` object to *brick* and list of `ursaRaster` objects to *stack*. S4 classes “*raster*”, “*brick*”, and “*stack*” are defined in package **raster**.

Usage

```
as.Raster(obj)

## S3 method for class 'ursaRaster'
as.Raster(obj)

## S3 method for class 'list'
as.Raster(obj)

## S3 method for class 'ursaStack'
as.Raster(obj)

## S3 method for class 'NULL'
as.Raster(obj)
```

Arguments

`obj` Object of class `ursaRaster` or list of `ursaRaster` objects

Details

Package **raster** is required for conversions.

The uppercase `as.Raster` is important, because `as.raster` is used in internal functions for coercion to object of class `raster`.

Single-banded `ursaRaster` object (with or without `colortable`) is coerced to `RasterLayer`. `Colortables` are kept.

Multi-banded `ursaRaster` object is coerced to `RasterBrick`. `Colortables` are destroyed.

Multi-layered object (list of `ursaRaster` objects) is coerced to `RasterStack`. `Colortables` are destroyed.

Value

Either `RasterLayer`, `RasterBrick`, or `RasterStack` object.

If package **raster** is not installed then return value is `NULL`

Note

Package **raster** is marked as "Suggested".

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
## test is skipped: raster's loading time is close to CRAN allowable test time
session_grid(NULL)
if (requireNamespace("raster")) {
  usedCRS <- ursa:::crsForceProj4()
  ursa:::crsForceProj4(TRUE) ## required for CRS comparison
  session_grid(regrid(mul=1/4))
  msk <- ursa_dummy(1,min=0,max=100)>40
  a1 <- ursa_dummy(1,min=200,max=500)[msk]
  a2 <- colorize(a1,ramp=FALSE,interval=FALSE,lazyload=FALSE)
  a3 <- as.integer(ursa_dummy(3,min=0,max=255.99))
  a4 <- ursa_stack(a3[msk])
  if (isLayer <- TRUE) {
    print(a1)
    r1 <- as.Raster(a1)
    message(as.character(class(r1)))
    print(r1)
    print(raster::spplot(r1))
    b1 <- as.ursa(r1)
    print(c(exported=a1,imported=b1,failed=b1-a1))
    print(c(theSameValue=identical(ursa_value(a1),ursa_value(b1))
      ,theSameGrid=identical(ursa_grid(a1),ursa_grid(b1))))
  }
  if (isLayerColortable <- TRUE) {
    r2 <- as.Raster(a2)
    message(as.character(class(r2)))
    print(r2)
    print(raster::spplot(r2))
    b2 <- as.ursa(r2)
    print(c(theSameValue=identical(ursa_value(a2),ursa_value(b2))
      ,theSameGrid=identical(ursa_grid(a2),ursa_grid(b2))))
  }
  if (isBrickOrRGB <- TRUE) {
    r3 <- as.Raster(a3)
    message(as.character(class(r3)))
    print(r3)
    print(raster::spplot(r3))
    raster::plotRGB(r3)
    b3 <- as.ursa(r3)
    print(c(theSameValue=identical(ursa_value(a3),ursa_value(b3))
      ,theSameGrid=identical(ursa_grid(a3),ursa_grid(b3))))
  }
  if (isStack <- TRUE) {
    r4 <- as.Raster(a4)
```

```

    message(as.character(class(r4)))
    print(r4)
    print(raster::spplot(r4))
    b4 <- as.ursa(r4)
    print(c(theSameValue=identical(ursa_value(a4),ursa_value(b4))
           ,theSameGrid=identical(ursa_grid(a4),ursa_grid(b4))))
  }
  ursa:::.crsForceProj4(usedCRS)
}

```

as.raster

Export raster object to a colored representation.

Description

as.raster transforms object of class ursaRaster to the object of class [raster](#) (package **grDevices**)

Usage

```
## S3 method for class 'ursaRaster'
as.raster(x, ...)
```

Arguments

x	ursaRaster object
...	Set of arguments, which are recognized via their names (using regular expressions) and classes:
	max number giving the maximum of the color values range. Passed to function as.raster for S3 class 'array'. Default is 255.

Value

A [raster](#) object. It is a matrix. The values of matrix are colors.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[as.array](#)

Examples

```

session_grid(NULL)
session_grid(regrid(mul=1/2))
a <- ursa_dummy(4,min=0,max=255)
a[a<70] <- NA
compose_open(layout=c(1,4),legend=NULL)
for (i in seq(4)) {
  panel_new()
  panel_plot(as.raster(a[seq(i)]),interpolate=FALSE)
  panel_annotation(paste("Number of channels:",i))
}
compose_close()

op <- par(mfrow=c(2,2),mar=rep(0.5,4))
plot(as.raster(a[1:1]),interpolate=FALSE)
plot(as.raster(a[1:2]),interpolate=FALSE)
plot(as.raster(a[1:3]),interpolate=FALSE)
plot(as.raster(a[1:4]),interpolate=FALSE)
par(op)

```

as.table	<i>Frequency of unique values</i>
----------	-----------------------------------

Description

as.table is an implementation of function `base::table` for values of raster image.

Usage

```

## S3 method for class 'ursaRaster'
as.table(x, ...)

ursa_table(x, ...)

```

Arguments

x ursaRaster object.

... Other arguments which passed to function `table` of package **base**.

Details

If ursaRaster has a `colortable`, then values are replaced by names of categories.

ursa_table is synonym to method as.table for class ``ursaRaster``.

Value

Object of class `table`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- colorize(pixelsize(),nbreak=4)
t1 <- as.table(a)
print(t1)
str(t1)
ursa_colortable(a) <- NULL
t2 <- as.table(a)
print(t2)
```

as.ursa	Create raster image from R objects or GDAL raster files.
---------	--

Description

as.ursa converts **R base** objects `matrix`, `array`, `numeric`, `data.frame` *list*, **sp** objects `SpatialGridDataFrame`, `SpatialPixelsDataFrame` and `SpatialPointsDataFrame`, **raster** objects `raster`, `stack` and `brick`, and GDAL raster files (using functions from **rgdal** package) to `ursaRaster` object.

Usage

```
as.ursa(obj, ... )
as_ursa(obj, ... )
```

Arguments

obj	R object for coercion
...	Depending on class of obj, arguments are passed to repsective functions.

Details

as_ursa is a synonym to as.ursa.
This is a high-level function to create `ursaRaster` objects. The follwed classes of R objects are implemented:

‘Data Class’	‘Appropriate method’
<code>array</code>	<code>ursa_new</code>
<code>matrix</code>	<code>ursa_new</code>
<code>numeric</code>	<code>ursa_new</code>
<code>data.frame</code>	<code>allocate</code>
<code>SpatialPointsDataFrame (sp)</code>	<code>allocate</code>
<code>SpatialPixelsDataFrame (sp)</code>	<code>allocate</code>
<code>SpatialGridDataFrame (sp)</code>	<code>ursa_new</code>

<code>list</code> of ursaRaster objects	<code>unlist</code>
<code>list</code> returned by <code>sf::gdal_read</code>	<code>ursa_new</code>
<code>list</code> (general)	Items \$x and \$y are required, If lengths of \$x and \$y are equal to dim of data, then
<code>ggmap</code> (ggmap)	<code>ursa_new.</code>
<code>raster</code> (raster)	<code>ursa_new.</code>
<code>brick</code> (raster)	<code>ursa_new.</code>
<code>stack</code> (raster)	<code>ursa_new.</code>
<code>bitmap</code> (magick)	<code>ursa_new.</code>
<code>character</code> (GDAL supported file name)	<code>read_gdal.</code>

Generally, `allocate` is used for objects with non-regular grid, and `ursa_new` is used for regular grids. The `raster grid` is defined from object properties or from `sessional grid`.

Color tables are supported for GDAL file names and **raster** objects (raster, brick, stack).

For ENVI *.hdr Labelled Raster Files there are alternatives:

1. Read object with GDAL (`read_gdal`);
2. Read object without GDAL (`read_envi`).

Value

Object of class ursaRaster

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a1 <- as.ursa(volcano)
print(a1)
display(a1)

session_grid(NULL)
b <- ursa_dummy(mul=1/16,bandname=format(Sys.Date()+seq(3)-1,"%A"))
print(b)

c1 <- b[[1]] ## equal to 'c1 <- as.matrix(b[1],coords=TRUE)'
str(c1)
b1a <- as.ursa(c1)
print(c(original=b[1],imported=b1a))
print(c(projection.b1a=ursa_proj(b1a)))
session_grid(NULL)
b1b <- as.ursa(c1$z)
print(b1b)
print(c(projection.b1b=ursa_proj(b1b)))

c2 <- as.data.frame(b)
str(c2)
session_grid(NULL)
```

```

b2a <- as.ursa(c2)
print(b2a)

session_grid(NULL)
attr(c2,"crs") <- NULL
b2b <- as.ursa(c2)
print(b2b)
print(ursa_grid(b2b))

c3 <- unclass(as.matrix(b,coords=TRUE))
str(c3)
session_grid(b)
b3a <- as.ursa(c3)
print(b3a)
print(ursa_grid(b3a))
session_grid(NULL)
b3b <- as.ursa(c3)
print(b3b)
print(ursa_grid(b3b))

c4 <- as.array(b)
str(c4)
session_grid(b)
b4a <- as.ursa(c4)
print(b4a)
print(ursa_grid(b4a))
session_grid(NULL)
b4b <- as.ursa(c4)
print(b4b)
print(ursa_grid(b4b))

n <- 20
c5 <- data.frame(y=runif(n,min=1000000,max=5000000)
                 ,x=runif(n,min=-3000000,max=1000000)
                 ,value=runif(n,min=0,max=10))
print(head(c5))
session_grid(b)
b5a <- as.ursa(c5)
print(b5a)
## to avoid over-timing during tests -- begin
  display(b5a)
## to avoid over-timing during tests -- end
session_grid(NULL)
b5b <- as.ursa(c5)
print(b5b)
## to avoid over-timing during tests -- begin
  display(b5b)
## to avoid over-timing during tests -- end

# b6 <- as.ursa(system.file("pictures/erdas_spnad83.tif",package="rgdal"))
b6 <- as.ursa(system.file("tif/geomatrix.tif",package="sf"))
print(b6)
display(b6,pal=c("black","white"),coast=FALSE,col="orange")

```

```

## package 'raster' is required -- begin
if (requireNamespace("raster")) {
  r <- raster::brick(system.file("external/rlogo.gri", package="raster"))
  print(r)
  b7 <- as.ursa(r)
  ursa_proj(b7) <- ""
  print(b7)
  display_rgb(b7)
}
## package 'raster' is required -- end

```

as_stars

*Raster coercion to 'stars'***Description**

Coercion from raster `ursaRaster` object to raster stars object defined in package **stars**.

Usage

```
as_stars(obj)
```

Arguments

`obj` Object of class `ursaRaster`.

Details

Simple coercion to stars object of package **stars**. Currently, color tables and attribution tables are not supported.

Value

Object of class `stars` for argument of class `ursaRaster`. Otherwise, `NULL`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```

session_grid(NULL)
a <- ursa_dummy(3)
x <- as_stars(a)
class(x)
if (requireNamespace("stars")) {
  print(x)
  b <- as_ursa(x)
  print(a)
}

```

```
    print(b)
}
```

bandname	<i>Band names for raster image.</i>
----------	-------------------------------------

Description

bandname (names) returns names of bands for object of class `ursaRaster` or existing ENVI labelled *.hdr file. bandname<- (names<-) sets names of bands for object of class `ursaRaster`.

Usage

```
bandname(x)
bandname(x) <- value

## S3 method for class 'ursaRaster'
names(x)

## S3 replacement method for class 'ursaRaster'
names(x) <- value
```

Arguments

x	Object of class <code>ursaRaster</code> . In the bandname function it is allowed to specify character 'ENVI labelled *.hdr' file name.
value	Character of length the same length of number of bands of x

Details

names is a synonym for bandname. names<- is a synonym for bandname<-

Value

For bandname and names, character vector.

For bandname<- and names<- , updated object of class `ursaRaster`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[nband](#)

Examples

```

session_grid(NULL)
a1 <- pixelsize()
a2 <- c("Band 1"=a1,Band2=a1/2,sqrt=sqrt(a1),NA)
print(a2)
print(bandname(a2))
bandname(a2)[1:2] <- c("Original","Half")
print(a2)
print(bandname(a2))

```

band_group

*Extract certain statistics of each band.***Description**

Function from this band.* list returns required statistics for each band.

Usage

```

band_mean(obj)
band_sd(obj)
band_sum(obj)
band_min(obj)
band_max(obj)
band_n(obj)
band_nNA(obj)
band_quantile(obj, ...)

```

Arguments

obj Object of class `ursaRaster`.

... Arguments, which are passed to generic `quantile` function, e.g. `probs`, `type`.

Details

- `band_mean` returns mean value.
- `band_sd` returns value of standard deviation with $n-1$ denominator.
- `band_sum` returns sum of values.
- `band_min` returns minimal value.
- `band_max` returns maximal value.
- `band_n` returns number of non-NA pixels.
- `band_nNA` returns number of NA pixels.
- `band_quantile` returns matrix of quantiles.

Value

Named vector of numerical or integer values. Band names are used for naming.

Note

Currently, implementation is not optimal, because firstly bundle of statistics is computed using [band_stat](#) function, and then required statistics is extracted.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[band_stat](#)

Examples

```
session_grid(NULL)
a <- ursa_dummy()
print(a)
print(a<80)
print(class(a))
a[a<80]
a[a<80] <- NA
b1 <- band_stat(a)
print(b1)
b2.n <- band_n(a)
str(b2.n)
b2.mean <- band_mean(a)
print(b1$mean)
print(b2.mean)
print(b1$mean-b2.mean)
print(band_quantile(a))
```

band_stat

Computes statistics for each band of raster.

Description

For each band of `ursaRaster` object, `band_stat` returns certain statistics (mean, sd, sum, min, max, number of non-NA pixels, number of NA pixels). Regarding to each band, it is *global* operations of map algebra.

Usage

```
band_stat(x, grid = FALSE, raw = FALSE)
```

Arguments

x	Object of class <code>ursaRaster</code> .
grid	Logical. If TRUE then metadata are returned instead of statistics. Default is FALSE
raw	Logical. For the case of raster values are categories, if <code>raw=TRUE</code> , then function returns statistics of categories; if <code>raw=FALSE</code> and names of categories can be transformed to numerical values, then function returns statistics for decategorized values. Default is FALSE.

Details

If raster values are not in memory or `grid=TRUE` then `ursa_info` is returned.

Generic function `print` for object of class `ursaRaster` uses returned value of `band_stat` function with formatted columns.

Statistics is computed for omitted NA values.

Value

`data.frame`. Row names are indices of bands. Column names are:

name	Band name.
mean	Mean value.
sd	Value of standard deviation with n-1 denomination.
sum	Sum of values.
min	Minimal value.
max	Maximal value.
n	Number of non-NA pixels.
nNA	Number of NA pixels.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

Columns extraction from returned data frame is in the group of `band.*` functions.

Examples

```
session_grid(NULL)
s <- substr(as.character(sessionInfo()),1,48)
a <- reclass(ursa_dummy(mul=1/2,bandname=s),ramp=FALSE)
band_stat(a,grid=TRUE)
b2 <- band_stat(a)
b3 <- band_stat(a,raw=TRUE)
str(b2)
str(b3)
```

```
print(b2)
print(a) ## 'print.ursaRaster' uses 'band_stat'
print(a,raw=TRUE)
```

blank

Does any band contain no information?

Description

Set of functions for checking is any or all bands have no data, and for retrieving indices for non-data bands.

Usage

```
band_blank(obj, ref = c("any", "0", "NA"), verbose = FALSE)
ursa_blank(obj, ref)
```

Arguments

obj	Object of class ursaRaster
ref	Character. Definition criteria, what is blank mean. If value "0", then blank is detected, if all values are 0. If value "NA", then blank is detected, if all values are NA. Default value is "NA": both NA and 0 are flags of blank. Non-character values are coerced to character.
verbose	Logical. Value TRUE provides progress bar. Default is FALSE.

Details

It is defined locally that if all values of band are NA or 0 (see description to argument ref), then such band is blank. The fact is [ursa_new](#) create new object in memory with default values NA, but [create_envi](#) writes zeros to disk quick. It is decided to consider both these cases as blank. Function band_blank checks blanks for each band of image. If all bands are blank then function ursa_blank returns TRUE.

Value

Function ursa_blank returns logical value of length 1.

Function band_blank returns logical value of length [nband\(obj\)](#).

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[is.na](#) returns object of class ursaRaster; it is mask of cells, which have NA value.

Examples

```
session_grid(NULL)
a <- ursa_new(bandname=c("first","second","third","fourth"))
ursa_value(a,"first") <- 0 ## 'a[1] <- 1' works, but it is slow
print(ursa_blank(a))
a[3] <- pixelsize()
a[4] <- a[3]>625
print(a)
print(band_blank(a))
print(which(band_blank(a)))
print(ursa_blank(a))
```

c

Combine bands into raster brick.

Description

This function is an instrument for appending bands or for reorganizing bands.

Usage

```
## S3 method for class 'ursaRaster'
c(...)
```

Arguments

... Objects of class `ursaRaster` or coerced to class `ursaRaster`. First argument should be the object of class `ursaRaster`. The objects in the sequence can be named.

Details

You may use this function to assign new bandname for single-band raster: `objDst <- c('Relative density'=objSrc)`

Use also 'Extract' operator `[ ]` to reorganize band sequence.

The returned object can be interpreted as a *brick* in the notation of package **raster**. To produce *stack* just call `list` or `ursa_stack`.

Value

`ursaRaster` object.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[ursa_brick](#) converts list of `ursaRaster` objects (*stack*) to a single multiband `ursaRaster` object (*brick*).

Examples

```
session_grid(NULL)
session_grid(regrid(mul=1/16))
a1 <- ursa_dummy(nband=2)
names(a1) <- weekdays(Sys.Date()+seq(length(a1))-1)
a2 <- ursa_dummy(nband=2)
names(a2) <- names(a1)
print(a1)
print(a2)
a3 <- a1[1]
print(names(a3))
a4 <- c(today=a3)
print(names(a4))
print(b1 <- c(a1,a2))
print(b2 <- c(a1=a1))
print(b3 <- c(a1=a1,a2=a2))
print(b5 <- c(a1=a1,a2=a2[1]))
print(b4 <- c(a1,'(tomorrow)'=a1[2])) ## raster append
print(b6 <- c(a1,50))
```

 chunk

Get indices for partial image reading/writing

Description

In the case of 'Cannot allocate vector of size ...' error message `chunk_band` returns list of bands indices, which are suitable for allocation in memory at once, `chunk_line` returns list of lines (rows) indices, which are suitable for allocation in memory at once. `chunk_expand` is used to expand lines indices and can be applied in focal functions.

Usage

```
chunk_band(obj, mem = 100, mul = 1, selected = NULL)
chunk_line(obj, mem = 100, mul = 1, selected = NULL)
chunk_expand(ind, size = 3)
```

Arguments

<code>obj</code>	Object of class <code>ursaRaster</code>
<code>mem</code>	Numeric. Memory size in MB, which is suitable for allocation.
<code>mul</code>	Numeric. Expansion or reduction factor (multiplier) of default value of memory allocation.

selected	Sequence of positive values or NULL. If specified (not NULL), then chunks are generated from this sequence of bands or lines. Default in NULL; all bands or lines are in sequence.
ind	Integer. Line indices.
size	Integer. Size of focal window.

Value

chunk_band returns list with sequences of bands

chunk_line returns list with sequences of lines

chunk_expand returns list:

src	expanded set if line indices
dst	matching of source indices in the expanded set

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
## 1. Prepare data
session_grid(NULL)
fname <- ursa:::maketmp(2)
a <- create_envi(fname[1],nband=3,ignorevalue=-99)
for (i in seq(nband(a)))
  a[i] <- pixelsize()^(1/i)
close(a)
rm(a)

## 2. Read
a <- open_envi(fname[1])
chB <- chunk_band(a,2)
str(chB)
for (i in chB)
  print(a[i])
chL <- chunk_line(a,2.5)
str(chL)
for (j in chL)
  print(a[,j])

## 3. Filtering with partial reading
b <- create_envi(a,fname[2])
fsize <- 15
for (j in chL) {
  k <- chunk_expand(j,fsize)
  b[,j] <- focal_mean(a[,k$src],size=fsize)[,k$dst]
}
d1 <- b[]
```

```
## 4. Filtering in memory
d2 <- focal_mean(a[],size=fsize)
close(a,b)
envi_remove(fname)
print(d1-d2)
print(round(d1-d2,4))
```

close

Close connections for files with data

Description

`close()` for `ursaRaster` object closes connection for opened file using inherited function `base::close`.
Function `close_envi()` closes opened connection for ENVI binary file.

Usage

```
## S3 method for class 'ursaRaster'
close(...)

close_envi(...)
```

Arguments

... Object or sequence of objects of class `ursaRaster`.

Value

NULL

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[close](#) of `base` package

Examples

```
session_grid(NULL)
a <- create_envi()
fname <- a$con$fname
message(paste("Created file",dQuote(basename(fname)),"will be deleted."))
print(dir(pattern=basename(envi_list(fname))))
close(a)
invisible(envi_remove(fname))
```

codec*Reduce and restore dimensions for sparse data matrix*

Description

compress reduces dimension of source image matrix and assigns indices. decompress uses indices for expansion of reduced image matrix.

Usage

```
decompress(obj)
compress(obj)
```

Arguments

obj Object of class ursaRaster

Details

After masking, vectorization of lines, points and small polygons image matrix is often sparse. Compressing (compress) is an option to reduce object size in memory. Decompressing (decompress) restore original data matrix.

Value

Object of class ursaRaster

Note

Currently, usage of compressed image matrix is limited. Spatial filtering (e.g. [focal_mean](#)) does not operate with compressed data.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
b <- as.data.frame(pixelsize())
b <- subset(b,x>1000000 & x<2000000 & y>3000000 & y<4000000)
a1 <- as.ursa(b)
print(a1)
print(object.size(a1))
a2 <- compress(a1)
print(a2)
print(object.size(a2))
a3 <- decompress(a2)
print(a3)
```

```
print(object.size(a3))
print(identical(a1,a3))
```

colorize

Create color table

Description

colorize assigns color table to raster image.

Usage

```
colorize(obj, value = NULL, breakvalue = NULL, name = NULL, pal = NULL, inv = NA,
         stretch = c("default", "linear", "equal", "mean", "positive",
                     "negative", "zero", "diff", "category", "julian",
                     "date", "time", "slope", "conc", "sd", "significance",
                     "bathy", "grayscale", "greyscale"),
         minvalue = NA, maxvalue = NA, byvalue = NA, ltail = NA, rtail = NA, tail = NA,
         ncolor = NA, nbreak = NA, interval = NA_integer_, ramp = FALSE, byte = FALSE,
         lazyload = TRUE, reset = FALSE, origin = "1970-01-01", format = "",
         alpha = "", colortable = NULL, verbose = FALSE, ...)
```

```
palettize(...) ## wrapper for non-spatial vectors
```

Arguments

obj	ursaRaster object or one-dimension numeric or character vector.
value	Numeric. Values to be assigned to categories.
breakvalue	Numeric. Values to be assigned to intervals.
name	Character. Names of categories.
pal	Function or character. If function then value should corresponded to function, which creates a vector of colors. If character then values should corresponded to R color names or hexadecimal string of the form "#RRGGBB" or "#RRGGBBAA".
inv	Logical. Invert sequence of colors.
stretch	Character. Either kind of value transformation ("linear", "equal") or pre-defined options with palette specification ("positive", "data", "significance", etc)
minvalue	Numeric. Lower range limit.
maxvalue	Numeric. Upper range limit.
byvalue	Numeric. Increment of the sequence from minvalue to maxvalue.
ltail	Numeric. Partition of omitted values at left tail.
rtail	Numeric. Partition of omitted values at right tail.

tail	Numeric. Partition of omitted values at both tail. If length of tail is 2 then left and right tails may differ.
ncolor	Numeric or interer. Number of desired colors (or categories).
nbreak	Numeric or interer. Number of desired separators between colors.
interval	Integer or logical. Logical is coerced to integer. How to underwrite categories? Value 0L means that for numeric data each color gradation corresponds to intervals but named as a single value (e.g., middle between low and high values). Value 1L means that values correspond to separators of color gradation. Value 2L is experimental for scales with zero. Default is NA: interval=0L for ramp=TRUE and interval=1L for ramp=FALSE.
ramp	Logical. Is color ramp required?
byte	Logical. Forcing to produce color table for storage in byte format (not more than 255 colors). Default is FALSE.
lazyload	Logical. If FALSE then raster is reclassified to categories. If TRUE then color table is created without any change to source raster and raster value just postponed for change. Default is TRUE.
reset	Logical. If TRUE and source raster has color table, then this color table is destroyed, and new one is created. Default is FALSE.
origin	Character. Origin for stretch="date" (passed to function as.Date) and stretch="time" (passed to function as.POSIXct). See description of origin in respective functions. Default is "1970-01-01".
format	Character. Format date/time objects for arguments stretch with values "date", "time", or "julian". Default is "" (character of length 0).
alpha	Character or numeric. The characteristics of transparency. If character, then hexadecimal values between "00" and "FF" are allowed, and then coerced to numeric value between 0 and 255. If numeric, and $0 \leq \alpha \leq 1$, then alpha is multiplied to 255. alpha=0 means full transparency, alpha=255 means full opacity. Default is ""; if palette has no alpha channel, then alpha is assign to "FF".
colortable	Object of class <code>ursaColorTable</code> or object of class <code>ursaRaster</code> with color table. Reference color table. Is specified, then all other arguments are ignored, expected lazyload. Default is NULL (unspecified).
verbose	Logical. Some output in console. Primarily for debug purposes.
...	For <code>colorize()</code> : If <code>pal</code> is a function, and argument names are in the format " <code>pal.*</code> " then prefix " <code>pal.</code> " is omitted, and the rest part is used for argument names, which are passed to <code>pal</code> function. For <code>palettize()</code> : Arguments, which are passed to <code>colorize()</code> .

Details

`palettize` is a wrapper `ursa_colortable(colorize(...))` to return color table for one-dimensional numeric or character vector.

`colortable` is designed to prepare pretty thematic maps.

Color ramping (ramp=TRUE) is not quick in computations and has no effective labelling. It is introduced to visualize non-thematic maps, and it is assumed that labeling can be omitted for such maps.

The labelling implementation is based on some improvements of `pretty` function. The notation of intervals is mixed by brackets and comparative symbols, for example: " ≤ 1.5 ", " $(1.5, 2.5]$ ", " $(2.5, 3.5]$ ", " > 3.5 "

Reserved values for interval:

- 0L or FALSE - no intervals. Values are interpreted as category, even if they are in non-nominal scale
 - 1L or TRUE - each category corresponds to interval. The low limit of lowest category is $-\text{Inf}$. The high limit of highest category is $+\text{Inf}$
 - 2L - different implementation of interval=1. In some cases may result more pretty labeling. If breaks is numerical vector and colors has zero length, then it is assumed interval scaling, and interval=1L is assigned to unspecified interval
- Finite values of extreme intervals are necessary sometimes, however this option is not implemented currently

Keywords for stretch to create pre-defined color tables:

- "positive" - lower limit is 0. Palette is "Oranges"
- "negative" - higher limit is 0. Palette is "Purples"
- "grayscale", "greyscale" - palette is "Greys". Usually used for raw satellite images.
- "mean" - designed for common thematic maps and for averaged map across set of maps. Palette is "Spectral"
- "sd" - designed for spatial mapping of standard deviation across set of maps. Palette is "Yl-GnBu"
- "diff" - diverge palette "RdBu". Absolute values of lower and upper limits are equal, zero is in the middle of palette. Designed for anomaly maps.
- "slope" - is similar to diff but without extreme colors, which are reserved for contouring of statistically significant areas.
- "significance" - designed to illustrate statistically significant areas of slope. The realisation is `colortable(obj, value=c(-0.999, -0.99, -0.95, -0.9, -0.5, +0.5, +0.9, +0.95, +0.99, +0.999), interval=1L, palname="RdBu")`
- "category" - Values are interpreted in nominal scale. Palette is based on random colors from "Pairs" palette.
- "conc" - designed for visualization of sea ice concentration data, which have lower limit 0 and higher limit 100. Palette is "Blues"
- "bathy" - designed for ocean depth (bathymetry) maps. Internally `colorize(obj, stretch="equal", interval=1L, palname="Blues", inv=TRUE)` is used to detect the crossing from shelf waters to deep water basin. Better practice is to do second step with manual specification of value argument.
- "internal" - continuous colors, designed for conversion to grayscale with keeping of intensities.
- "default" - allowing to detect stretch by intuition, without any strong mathematical criteria

It is allowed manual correction of labels using followed code example: `names(ursa_colortable(x)) <- c("a<=0", "0<a<=1", "a>1")`

Value

Object of class `ursaRaster` with named character vector of item `$colortable`

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[ursa_colortable](#), [ursa_colortable<-](#)

Examples

```
session_grid(NULL)
a <- pixelsize()-350
print(a)
b1 <- colorize(a,ramp=FALSE)
print(ursa_colortable(b1))
b2 <- colorize(a,interval=1,stretch="positive",ramp=FALSE)
print(ursa_colortable(b2))
b3 <- colorize(a,interval=2,stretch="positive",ramp=FALSE)
print(ursa_colortable(b3))
b4 <- colorize(a,value=c(150,250),interval=1)
print(ursa_colortable(b4))
names(ursa_colortable(b4)) <- c("x<=150","150<x<=250","x>250")
print(ursa_colortable(b4))
display(b4)
```

colortable

Color Tables of raster images.

Description

Manipulation with color tables of raster images.

Usage

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

```
## S3 method for class 'ursaColorTable'
x[i]
```

```
ursa_colortable(x)
```

```
ursa_colortable(x) <- value
```

```

ursa_colorindex(ct)

ursa_color(ct, ...)

category_value(ct, name)

## S3 method for class 'ursaColorTable'
names(x)

## S3 replacement method for class 'ursaColorTable'
names(x) <- value

```

Arguments

<code>x</code>	ursaRaster object. Extended for numeric (integer or real) of character vector.
<code>ct</code>	ursaColorTable object with or without indexing.
<code>value</code>	Named character vector. In Replacement functions: For <code>ursa_colortable()</code> : values are colors in “#RRGGBB” notation or R color names (colors). <code>names(value)</code> are names of categories. For <code>names()</code> : values are names of categories. If length of names is $n-1$, where n is length of colors, then intervaling is assumed, and value are assign to interval breaks.
<code>i</code>	Integer vector. Indices specifying elements to extract part (subset) of color table.
<code>name</code>	Character of length one. Name of category or name of colortable to get desired integer value stored in data.
<code>...</code>	In <code>print()</code> , passing to generic print . Currently not used. In <code>ursa_colortable()</code> , passing to generic print . Currently not used. In <code>ursa_color()</code> , passing to colorize .

Details

The example of the class structure

```

Class 'ursaColorTable'  Named chr [1:4] "#313695" "#BCE1EE" "#FDBE70" "#A50026"
..- attr(*, "names")= chr [1:4] "<= 450" "(450;550]" "(550;650]" "> 650"

```

It is recommended to use `ursa_colortable` and `ursa_colortable<-` instead of `colortable` and `colortable<-`. `ursa_colortable` and `colortable` are synonyms. `ursa_colortable<-` and `colortable<-` are synonyms too. Package **raster** contains [colortable](#) and [colortable<-](#) functions. `colortable` and `colortable<-` will be remove from this package if the case of frequent joint use of both packages.

If color tables describe continuous and non-intersecting intervals, then `print` gives additional line of extracted breaks.


```

print(ursa_colortable(b2))
names(ursa_colortable(b2))[4] <- "Polar"
print(ursa_colortable(b2))
display(b2)

lab <- sample(c("A", "B", "C"), 9, replace=TRUE)
lab
ct <- ursa_color(lab)
names(ct) <- lab
ct
category_value(b2, "High")

```

commonGeneric

Some generic functions for ursaRaster class.

Description

Set of generic functions, implemented for objects of ursaRaster class.

Usage

```

## S3 method for class 'ursaRaster'
duplicated(x, incomparables = FALSE, MARGIN = 2, fromLast = FALSE, ...)

## S3 method for class 'ursaRaster'
diff(x, lag = 1, differences = 1, ...)

```

Arguments

x	Object of ursaRaster class
incomparables	Passed to S3 method duplicated for class matrix.
MARGIN	Overwritten to value 2. Passed to S3 method duplicated for class matrix.
fromLast	Passed to S3 method duplicated for class matrix.
lag	Passed to default S3 method diff.
differences	Passed to default S3 method diff.
...	Other arguments, which are passed to the respective S3 method.

Value

duplicated(): logical of length equal to number of bands.
diff(): ursaRaster object.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[duplicated](#), [diff](#),

Examples

```
a <- ursa_dummy(5)
a[3] <- a[2]
a
duplicated(a)
diff(a)
```

compose_close	<i>Finish plotting</i>
---------------	------------------------

Description

Function `compose_close` does followed tasks: 1) completes all unfinished actions before shutting down graphical device, 2) cuts extra margins, and 3) opens resulted PNG file in the associated viewer.

Usage

```
compose_close(...)

## non-public
.compose_close(kind = c("crop2", "crop", "nocrop"),
               border = 5, bpp = 0, execute = TRUE, verbose = FALSE)
```

Arguments

... Set of arguments, which are recognized via their names and classes, and then passed to `.compose_close`:

	Pattern (<code>compose_close</code>)	Argument (<code>.compose_close</code>)
	(<code>^\$ crop kind</code>)	<code>kind</code>
	(<code>border frame</code>)	<code>border</code>
	<code>bpp</code>	<code>bpp</code>
	(<code>render execute view open</code>)	<code>execute</code>
	<code>verb(ose)*</code>	<code>verbose</code>
<code>kind</code>	Character keyword for cutting of excess white spaces. If <code>kind="nocrop"</code> then there is no cut. If <code>kind="crop"</code> then only outer margins are cutted. If <code>kind="crop2"</code> then all outer margins and inner white spaces (<i>e.g.</i> , between color bar panel and text caption) are cutted.	
<code>border</code>	Non-negative integer. Number of pixels for margins, which are not cropped. Default is 5L.	

bpp	Integer. Bits per pixel for output PNG file. Valid values are 0L, 8L, 24L. If bpp=0L, then 8 bpp is used for "windows" type of PNG device, and 24 bpp is used for "cairo" type of PNG device. The type of device is specified in compose_open function.
execute	Logical. Should created PNG file be opened in the associated external program for viewing graphical files? Default is TRUE.
verbose	Logical. Value TRUE provides some additional information on console. Default is FALSE.

Details

The cut manipulations (crop="crop" or crop="crop2") are implemented using [readPNG](#) and [writePNG](#) functions of package **png**. These functions have limitations in the memory allocation.

Function `compose_close` clears all internal graphical options, specified during [compose_open](#) executing.

Some parameters are specified in [compose_open](#): weather output PNG file will be removed after opening (logical `deafter`), or what is the time of waiting for file opening and next removing (numerical wait in seconds).

Value

Function returns NULL value.

Warning

Currently, `execute=TRUE` is implemented for Windows platform only using construction `R CMD open \emph{fileout}`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- ursa_dummy(nband=6,min=0,max=255,mul=1/4)

## exam 1
compose_open()
compose_close()

## exam 2
compose_open(a)
compose_close()

## exam 3
compose_open("rgb",fileout="tmp1")
compose_plot(a[1:3])
compose_close(execute=FALSE)
Sys.sleep(1)
a <- dir(pattern="tmp1.png")
```

```
print(a)
file.remove(a)
```

compose_design	<i>Organize multi-panel layout with images and color bars.</i>
----------------	--

Description

compose_design prepares scheme for layout of images and color bars.

Usage

```
compose_design(...)
```

Arguments

- ... Set of arguments, which are recognized via their names and classes:
- obj** Object of class `ursaRaster` or list of objects of class `ursaRaster` or `NULL`. Default is `NULL`. Used to detect panel layout and coordinate reference system.
- layout** Integer of length 2, integer of length 1, two-dimensional matrix or `NA`. Layout matrix has dimensions `c(nr, nc)`, where `nr` is number of rows, and `nc` is number of columns. If `layout` is positive integer of length 1, then sequence of this value unfolds to layout matrix using argument `ratio`. If `layout=NA` then layout matrix is recognized internally using number of bands of `obj` and argument `ratio`. If `layout=NA` and `obj=NULL` then matrix `c(1,1)` is used.
- byrow** Logical. The order of filling of layout matrix. Default is `TRUE`. If `byrow=TRUE` then matrix is filled by rows (from top row, consequently from left element to right element, then next row). If `byrow=FALSE` then matrix is filled by columns.
- skip** Positive integer of variable length. Default is `NULL` (length is zero). Indices of panels in the layout matrix, which are not used.
- legend** The description of rules how color bars (legends) or panel captions are located in the layout. It is the list of embedded lists of two elements, which describe the color bars position in the layout. If Default is `NA`, it means using of internal rules. If `legend=NULL` then no plotting of color bars. If `legend` is positive integer in the range `1L:4L`, then single color bar is used and legend's side is corresponded to margins of `R` graphic system.
- side** Positive integer `1L, 2L, 3L, or 4L`. Default is `NA`. Simplification of color bar position in the case that single color bar is used. The value is corresponded to margins of `R` graphic system. The synonym of integer value of `legend`.
- ratio** Positive numeric. The desired ratio of layout sides (width per height). If `layout=NA` then the dimensions of layout matrix are defined internally to get the given ratio of layout's width per height. The default is $(16+1)/(9+1)$ in the assumption of optimal filling on the usual 16:9 screens.

fixed Logical. If TRUE then it is assuming that layout will have the same proportions as sessional grid sizes (rows and columns). For this case, argument ratio is reassigned as a desired ratio (width per height) for single panel. Default is FALSE.

Details

Function `compose_design` extracts and validates required arguments from a list of parameters (three-dots construct) and passes them to internal function `.compose_design`.

Argument `legend` is a `list` or coerced to a `list`. The length of this list is equal to number of color bars; each item describes certain color bar. This description is a list again with two elements, which describes the position of color bar in relation to main panels of images.

If argument `legend` is in interval 1L:4L then it is interpreted as argument `side` in functions `axis`, `mtext`. Argument `side` in function `compose_design` plays the same role. It is introduced for consistency with R graphic system.

In the one of example below (See *Examples* section) the layout with dimension of two rows by three columns is considered (`layout=c(2,3)`). The dimension of resulting layout matrix is `c(7,9)`, where $7=2*2+3$, and $9=3*2+3$.

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	0	0	0	0	0	0	0	0	0
[2,]	0	0	0	0	0	0	0	0	0
[3,]	0	0	1	0	2	0	3	0	0
[4,]	0	0	0	0	0	0	0	0	0
[5,]	0	0	4	0	5	0	6	0	0
[6,]	0	0	0	0	0	0	0	0	0
[7,]	0	0	0	0	0	0	0	0	0

The complicated color bar structure is specified via R's `list` function:

```
> leg <- list("7"=list(row=1,col=0),"8"=list(2,"left")
+           ,"9"=list("full","right"),"10"=list("top","full")
+           ,"11"=list(99,1:2),"12"=list("bottom",3))
> str(leg)
$ 7 :List of 2
..$ row: num 1
..$ col: num 0
$ 8 :List of 2
..$ : num 2
..$ : chr "left"
$ 9 :List of 2
..$ : chr "full"
..$ : chr "right"
$ 10:List of 2
..$ : chr "top"
..$ : chr "full"
$ 11:List of 2
..$ : num 99
```



```

..$ : int [1:2] 1 2
$ 12:List of 2
..$ : chr "bottom"
..$ : num 3

```

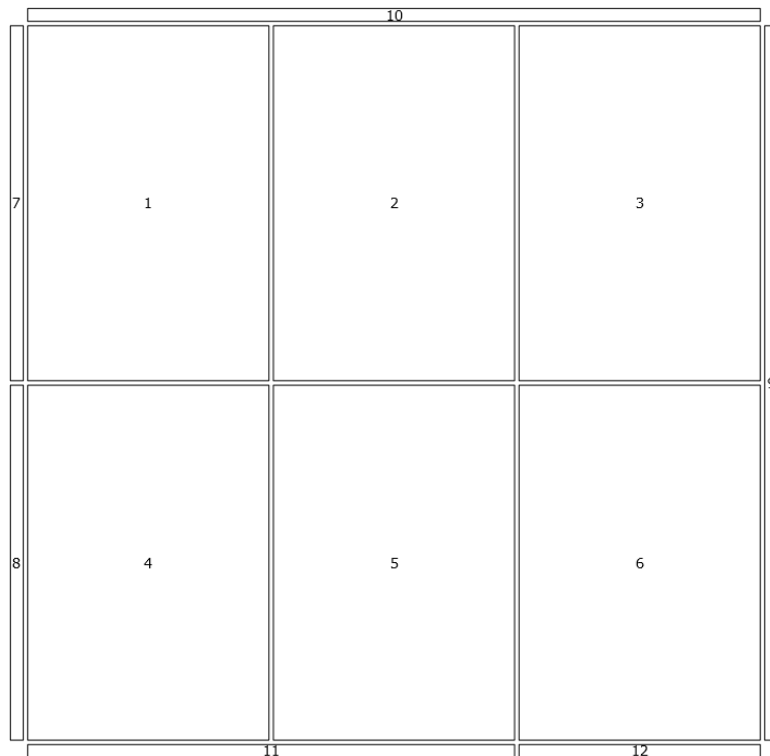
Here, six color bars are specified. It is a list of six lists (sub-lists). First item of sub-list is row number, and the second one is column number. Integers can be replaced by character keywords.

For row-position, "top" means 0L (less than first row), "bottom" means large integer value (greater than last row, currently, 99L), "first" means 1L, "last" means number of last row (2L in this example), "full" means whole range from first to last rows (1L:2L in this example). Values "top" and "bottom" are used for horizontal color bars (last three sub-lists), and the rest for vertical color bars (first three sub-lists).

For column-position, "left" means 0L (less than first column), "bottom" means large integer value (greater than last column, currently, 99L), "first" means first column (1L), "last" means last column (3L in this example), "full" means whole range from first to last columns ((1L:3L in this example)). Values "left" and "right" are used for vertical color bars, and the rest are for horizontal ones.

The resulting layout is a sparse matrix with zero values for each even row and each column. These zeros play the role of white space between panels in the plotted layout. In our example, values 1L:6L are corresponded to six map panels, and values 7L:12L are corresponded to six narrow panels of color bars (legends).

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	0	0	10	0	10	0	10	0	0
[2,]	0	0	0	0	0	0	0	0	0
[3,]	7	0	1	0	2	0	3	0	9
[4,]	0	0	0	0	0	0	0	0	0
[5,]	8	0	4	0	5	0	6	0	9
[6,]	0	0	0	0	0	0	0	0	0
[7,]	0	0	11	0	11	0	12	0	0



Value

It is a list of class `ursaLayout`.

layout	Integer matrix with dimension $c(2*nr+3, 2*nc+3)$, where <code>nr</code> and <code>nc</code> are number of rows and columns of the layout matrix. The layout matrix of image panels is surrounded by colorbar panels. The original layout matrix is expanded by adding zero columns and rows. In the new matrix each even column has zero values, and each even row has zero values.
image	Nonnegative integer. Number of panels with images.
dim	Nonnegative integer of length two. Number of rows and number of columns for panel layout.
legend	Nonnegative integer. Number of panels with color bars (legends).

The returned value is passed to function `compose_open` and further is kept in the `options` until calling of `compose_close`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
```

```

a <- ursa_dummy(nband=5,min=1,max=200,mul=1/8)
b <- list(colorize(a[1:3],pal.rich=240,pal.rotate=0)
          ,colorize(sqrt(a[4:5]),pal.rich=-15,pal.rotate=0,stretch="equal"))

cl1 <- compose_design(layout=c(2,3),byrow=TRUE,legend=NULL)
print(cl1)
compose_open(cl1)
compose_close()

cl2 <- compose_design(layout=c(2,3),byrow=FALSE,legend="left")
print(cl2$layout)
compose_open(cl2)
compose_close()

cl3 <- compose_design(a,side=2)
print(cl3)
compose_open(cl3)
compose_close()

cl4 <- compose_design(b)
print(cl4)
## to avoid over-time during example check -- begin
compose_open(cl4)
compose_plot(b,decor=FALSE,las=2)
compose_close("nocrop")
## to avoid over-time during example check -- end

cl5 <- compose_design(b,byrow=FALSE,skip=3
                      ,legend=list(list("full","left"),list(1:2,"right")))
compose_open(cl5)
compose_plot(b,decor=FALSE)
compose_close("nocrop")

leg <- list(list(1,0),list(2,"left")
            ,list("full","right"),list("top","full")
            ,list(99,1:2),list("bottom",3))
str(leg)
cl6 <- compose_design(layout=c(2,3),skip=NA,legend=leg)
print(cl6)
compose_open(cl6,scale=3,pointsize=16)
compose_close("nocrop")

cl7 <- compose_design(layout=matrix(c(1,1,3,2,2,0),nrow=2,byrow=TRUE))
print(cl7)
compose_open(cl7)
compose_close()

```

Description

compose_legend recognizes color tables and characters among arguments and passes them to suitable functions for plotting on margins outside of panel area.

Usage

```
compose_legend(...)
```

Arguments

... If first argument is a list, then either `ursaColorTable` or `character` objects are detected in this list. `ursaColorTable` can be extracted from `ursaRaster` (if presents). Other objects are coerced to character.

If first argument is `ursaColorTable` or `ursaRaster` with color tables, then other arguments are interpreted as color tables. If coercion to color table is impossible, the coercion is to character.

`legend_colorbar` is called for objects of class `ursaColorTable`. `legend_mtext` is called for objects of class `ursaColorTable`. If first argument is a list, then other arguments are passed to respective function calls.

Details

Named list in the first argument is allowed or named vectors are allowed if first argument is not a list. For `legend_colorbar` name of object can be used as an argument units.

This function is designed to make plot on moderate level of usage with the followed construction:

```
compose_open(...)  
compose_panel(...)  
compose_legend(...)  
compose_close(...)
```

Function `compose_panel` returns list of color tables of plotted rasters, and followed sequence is available:

```
ct <- compose_panel(a)  
compose_legend(ct) # or, if 'a' has color tables, then 'compose_legend(a)'
```

Value

NULL

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

`legend_colorbar`
`legend_mtext`

Examples

```
session_grid(NULL)
b <- lapply(as.list(ursa_dummy(2)),colorize)
cd <- compose_design(layout=c(1,2),legend=list(list(1,"left"),list(1,"right")
                                                ,list("top","full"),list("bottom",1)))

for (i in 1:4) {
  compose_open(cd,dev=i==1)
  ct <- compose_panel(b,decor=FALSE)
  if (i==2)
    compose_legend(ct)
  else if (i==3)
    compose_legend(ct[[1]],'Tomorrow'=b[[2]]
                  ,top="This is example of legend composition"
                  ,format(Sys.Date(),"(c) %Y"))
  else if (i==4)
    compose_legend(c(ct,"top","bottom"),units=c("left","right"))
  compose_close()
}
```

compose_open

Start displaying

Description

compose_open create plot layout and open PNG graphic device.

Usage

```
compose_open(...)
```

Arguments

- ...
- Set of arguments, which are recognized via their names and classes:
- mosaic** Layout matrix or reference object to produce layout matrix. It is permitted to do not use name for this argument. Multiple types of argument are allowed: 1) object of class `ursaRaster`, 2) [list](#) of `ursaRaster` objects (raster stack), 3) object of class `ursaLayout` from function [compose_design](#), 4) character keyword or 5) missing. Default is NA.
 - fileout** Character. Name for created output file. Supported PNG (*.png), JPEG (*.jpg), WEBP (*.webp) and SVG (*.svg) extensions. Missed extension assumes PNG. If absent ("") then temporal file is created and removed in wait seconds after opening in the associated external viewer. Default is absent ("").
 - dpi** Positive integer. Dots (or pixels) per inch (DPI). The nominal resolution of created output (default PNG) file. Default is 96L. The same as `res` argument in the [png](#) function.

- pointsize** Positive integer. The pointsize of plotted text as it is applied in the [png](#) function. Default is NA. If pointsize=NA then it is taken value 16L multiplied to relative DPI (dpi/96). In the case of unspecified scale and pointsize the size of text is defined internally.
- scale** Positive numeric or character. The scale factor applied to dimensions of original raster. Default is NA. If scale is unspecified (scale=NA), then scale is defined internally for *intuitively* better fitting in HD, FHD displays (single-panelled layout 900x700). If scale is character (e.g., "8000000", "1:8000000") then dimensions of image panels are defined using "one centimeter of map is corresponded to 8000000 centimeres of site" rule.
- width** Positive numeric or character. The desired width of each panel of multipanel layout. If width is numeric, then units are pixels. If width is character (e.g., "12.5", "12.5 cm", "12.5cm") then units are centimeters in agreement with dpi argument. Default is NA. If width is unspecified (width=NA) then value 900 is used for single panelled layout.
- height** Positive numeric or character. The desired height of each panel of multipanel layout. If height is numeric, then units are pixels. If height is character (e.g., "9.6", "9.6 cm", "9.6cm") then units are centimeters in agreement with dpi argument. Default is NA. If height is unspecified (height=NA) then value 700 is used for single panelled layout.
- colorbar** **or** **colorbarwidth** Positive numeric. Scale factor to increase (colorbar>1) or decrease (colorbar<1) width (the shortest dimension) of color bars (legends). Default value (NA) means 2.8% of image panel width.
- indent** **or** **space** **or** **offset**. Positive numeric. Scale factor to increase (space>1) or decrease (space<1) the white space between image panels and between image and color bar panels. Default value (NA) means 0.8% of image panel width.
- box** Logical. If TRUE then boundary box is plotted around image panels and color bar panels. It is a transparent rectangle with black border. Default is TRUE.
- deafter** Logical. If TRUE then created PNG file will be deleted after viewing. Default is FALSE for specified file names and TRUE for unspecified (temporal) file names. It is implemented as file removing after opening in the external PNG viewer.
- wait** Positive numeric. Seconds between PNG file opening in the associated program and file removing. It make sense only if deafter=TRUE. Default is 1.0 (one second).
- device** **or** **type** Character keyword, either "cairo", "windows" or "CairoPNG" for OS Windows, and either "cairo", "cairo-png", "Xlib" or "quartz" for other OSes. Should be plotting be done using cairographics or Windows GDI? The same as type argument in the [png](#) function, excepting "CairoPNG", which is handed by **Cairo** package. Default is "cairo".
- antialias** Character keyword, either "none" or "default". Defines the effect on fonts. The same as antialias argument in the [png](#) function. Default is "default".
- font** **or** **family** A length-one character vector. Specifies the font family. The same as family argument in the [png](#) function. Default is "sans" for device="windows"

and "Tahoma" for device="cairo".

bg or **background** Character. The background color in PNG file. Passed as argument bg to [png](#) function. Default is "white"

retina Positive numeric. Scale coefficient for retina displays. Default is taken from `getOption("ursaRetina")`; if it missed, then 1.

dev Logical. If TRUE then this **developer** tool shows created layout without any the followed plot functions from this package are ignored. Default is FALSE

verbose Logical. Shows additional output information in console. Default is FALSE.

... Arguments, which can be passed to [compose_design](#) function.

Details

Other usage of `compose_open(..., dev=TRUE)` is

```
compose_open(..., dev=FALSE)
compose_close()
```

The reason to use [compose_design](#) function before `compose_open` is to reduce number of arguments in the case of complicated layout matrix and non-standard settings.

`compose_open` passes arguments to [png](#) function.

If character values are specified for arguments width, height or scale, then layout development is oriented to produce PNG file, which will be used as a paper copy. Character values for width and height are in centimeters. Character value V or 1:V of scale defines scale 1/V.

The Cairo device (device="cairo") is more quick on MS Windows computers. However Windows GDI may produce less depth of colors (even 8 BPP) in the case of no font antialiasing. Usage of Windows GDI (device="windows") is a way to produce illustrations for scientific journals with strict requirements of minimal line width, font size, *etc.*

The PNG layout reserves extra margins for captions of color bars. These margins are filled by white spaces. The cropping of layout applies to created PNG file using read-write functions of package **png**. Only white ("white", "#FFFFFF") or transparent ("transparent") colors are recognized as white spaces. Therefore, specification of `bg!="white"` or `bg!="transparent"` breaks PNG image cropping.

It is noted that Cyrillics is supported on Windows GDI (device="windows") and is not supported on Cairo (device="cairo") types of PNG device on MS Windows platform.

Argument retina is ignored for leaflet-compatible tiling.

Value

Name of created PNG file.

If dev=TRUE then output on console is layout matrix.

The set of required parameters for plotting are kept until function [compose_close](#) call via [options](#).

ursaPngAuto For developers. Indicator of high-level functions for internal use (manual set; value is TRUE). Or, can be missed.

ursaPngBox	Argument box. If TRUE then box is called for each panel of layout at the end of plotting.
ursaPngDelafter	Argument delafter. Applied in the function compose_close .
ursaPngDevice	Argument device. Applied for effective plotting of rasters and checking the ability for final reducing color depth from 24 to 8 bpp.
ursaPngDpi	Argument dpi. Currently used for verbose only.
ursaPngFamily	Applied for text plotting in annotations and legends.
ursaPngFigure	Set 0L. Specifies number of current panel in layout matrix. Used to detect term for applying \$ursaPngBox option.
ursaPngFileout	Name of created PNG file.
ursaPngLayout	Layout matrix, the object of class ursaLayout .
ursaPngPaperScale	Numeric. Used for scalebar representation on the paper-based maps. If value 0, then scalebar is display-based. If value is greater 0 then the scale is exact. If value is -1 then the resonable rounding is used for scale displaying.
ursaPngPlot	The opposite to argument dev. If \$ursaPngPlot is FALSE then any plotting functions of this package are ignored.
ursaPngScale	The actual value of argument scale, specified during function call or stated internally.
ursaPngShadow	Set "". Used for shadowing of the part of color bars in the case of semi-transparent land or ocean filling mask in the panel_coastline function.
ursaPngSkipLegend	Integer vector of non-negative length. Defines list of images panels, for which the color bars are not displayed.
ursaPngWaitBeforeRemove	Argument wait. Applied in the function compose_close for temporal PNG file.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
b <- ursa_dummy(nband=4,min=0,max=50,mul=1/4,elements=16)
p <- list(colorize(b[1:2],pal.rich=240,pal.rotate=0)
          ,colorize(sqrt(b[3:4]),pal.rich=-15,pal.rotate=0,stretch="equal"))
p

## exam #01
compose_open(width=950,dpi=150,pointsize=16,legend=NULL,dev=TRUE)

## exam #02
compose_open(pointsize=8,dpi=150,scale="1:130000000")
compose_plot(colorize(b[1]),scalebar=TRUE,coast=FALSE)
```



```

compose_close()

## exam #03
cl <- compose_design(layout=c(2,4)
                     ,legend=list(list("top","full"),list("bottom",1:3)))
compose_open(cl,dev=TRUE)

## exam #04
cl <- compose_design(p,layout=c(2,3),skip=c(2,4,6))
compose_open(cl,dev=TRUE)

## exam #05
cl <- compose_design(p,side=3)
compose_open(cl,dev=FALSE,bg="transparent")
compose_close()

```

compose_panel

Plot raster images and decorations on the multipanel layout.

Description

compose_panel divides the multi-band raster image (*brick*) or layers of raster images (*stack*) on the sequence of single-band images and plots each image on the separate panel of layout. Panel plotting is finalized by adding of decoration (gridlines, coastline, annotation, scalebar).

Usage

```
compose_panel(..., silent = FALSE)
```

Arguments

...	Set of arguments, which are passed to panel_new , panel_raster , panel_coastline , panel_graticule , panel_annotation , panel_scalebar .
silent	Logical. Value TRUE cancels progress bar. Default is FALSE.

Details

For each panel of layout the sequence of called functions is permanent:

```
panel_new --> panel_raster --> panel_coastline --> panel_graticule --> panel_annotation
--> panel_scalebar.
```

If this order is undesirable, then call these functions in the required sequence.

Value

NULL

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also[compose_plot](#), [compose_legend](#)[panel_new](#), [panel_raster](#), [panel_coastline](#), [panel_graticule](#), [panel_annotation](#), [panel_scalebar](#)**Examples**

```

session_grid(NULL)
a <- ursa_dummy(6)
b1 <- list(maxi=a[1:4]*1e2,mini=a[5:6]/1e2)
print(b1)
b2 <- lapply(b1,function(x) colorize(x,nbreak=ifelse(global_mean(x)<100,5,NA)))
compose_open(b2,byrow=FALSE
             ,legend=list(list("bottom",1:2),list("bottom",3),list("left")))
ct <- compose_panel(b2,scalebar=2,coastline=3:4,gridline=5:6,gridline.margin=5
                  ,annotation.text=as.character(seq(6)))
compose_legend(ct)
legend_mtext(as.expression(substitute(italic("Colorbars are on the bottom"))))
compose_close()

```

compose_plot

*Plot layout of images and color bars.***Description**

compose_plot plots images (raster brick or raster stack) and corresponding color bars according to given rectangular layout.

Usage

```
compose_plot(...)
```

Arguments

... Set of arguments, which are passed to [compose_panel](#) and [compose_legend](#)

Details

Function merges to functions. The first one plots image layout and returns list of color tables. The second one plots legend (colorbars) based on returned color tables. Simplified description is:

```

ct <- compose_panel(...)
compose_legend(ct,...)

```

These two functions are separated to allow use additional plotting on image panel after primary plot of raster and decorations before panel change or legend plot.

Value

This function returns NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[compose_panel](#)
[compose_legend](#)

Examples

```
session_grid(NULL)
a <- ursa_dummy(nband=6,min=0,max=255,mul=1/4)
if (example1 <- TRUE) {
  b1 <- ursa_brick(a)
  # b1 <- colorize(b1,stretch="positive",ramp=FALSE)
  compose_open(b1)
  compose_plot(b1,grid=FALSE,coast=FALSE,scale=FALSE,trim=1
               ,stretch="positive",ramp=!FALSE)
  compose_close()
}
if (example2 <- TRUE) {
  b2 <- ursa_stack(a)
  compose_open(b2)
  compose_plot(b2,grid=FALSE,coast=FALSE,labels=5,trim=2,las=0)
  compose_close()
}
```

create_envi

Create ENVI or GDAL files on disk

Description

create_envi creates ENVI binary and header files on disk. ENVI binary file is filled by blank (zero) values.

create_gdal is just an entry for GDAL wrapper; currently via internal ENVI implementation.

Usage

```
create_gdal(x, ...)
```

```
create_envi(x, ...)
```

Arguments

x	Filename, or any refenerce object to help assign properties of new ENVI file. Can be missed.
...	Use name = value sequence. Properties of new ENVI file are extracted from keywords in 'name' and data types of 'value'.

Details

Prior **ursa** version < 3.10, `create_gdal()` used classes and methods from package **rgdal**. Currently, alternatives are not found for complete replacement of **rgdal**. At the present, ENVI binary and header are created, firstly, and `close()` transforms to desired GDAL format, finally.

`create_envi` and `create_gdal` use parameters of grid (boundary box, cell size, projection) from reference object of class `ursaRaster` in argument `x` or calls `session_grid`. You may specify values of GDAL or ENVI binary file later using `[<-`. If `x` is object of class `ursaRaster` then metadata parameters (interleave, data type, ignore value, etc) are inherited.

Keywords:

- `fname` - character. File name for created GDAL or ENVI file.

For create_envi only: If compress of connections is not specified then example for “file-out” file name:

- “fileout” - If external ‘gzip’ is found then “fileout.envigz” is created else “fileout.envi”
- “fileout.envi” - “fileout.envi” is created without any compression.
- “fileout.” - “fileout” is created without any compression.
- “fileout.bin” - “fileout.bin” is created without any compression.
- “fileout.img” - “fileout.img” is created without any compression.
- “fileout.dat” - “fileout.dat” is created without any compression.
- `driver` - character. *For create_gdal only.* Which GDAL driver is used.
- `layername` - character of length>=1. Layernames (‘Band name’ in ENVI header file)
- `bandname` - character of length>=1. Layernames (‘Band name’ in ENVI header file)
- `name` - character of length>=1. Layernames (‘Band name’ in ENVI header file)
- `nodata` - integer or numeric. Value in GDAL or ENVI binary file, which is interpreted as NA in R
- `ignore` - integer or numeric. Value in GDAL or ENVI binary file, which is interpreted as NA in R
- `ignorevalue` - integer or numeric. Value in GDAL or ENVI binary file, which is interpreted as NA in R
- `bg` - integer or numeric. Value in GDAL or ENVI binary file, which is interpreted as NA in R
- `connection` - character. *For create_envi only.* `connections` for ENVI binary file.

Valid values are:

- “gz” - connection is “gzfile”
- “bz” - connection is “bzfile”
- “xz” - connection is “xzfile”
- “file” - connection is “file”
- `interleave` - character. Interleave. Valid values are “bsq”, “bil”, “bip”. For `create_gdal` and `driver="GTiff"` valid values are “bsq” and “bil”.
- `datatype` - character or integer (numeric). Data type.

Valid values are:

- 1, "byte", "Byte", "UInt8" = Byte: 8-bit unsigned integer
- 2, "integer", "Int16" = Integer: 16-bit signed integer
- 3, "Int32" = Long: 32-bit signed integer
- 4, "real", "float", "Float32" = Floating-point: 32-bit single-precision
- 5, "Float64" = Double-precision: 64-bit double-precision floating-point
- 11, "UInt8" = Byte: 8-bit signed integer. **Not in specification.** Only for use with this package.
- 12, "UInt16" = Integer: 16-bit unsigned integer
- 13, "UInt32" = Long: 32-bit unsigned integer

Specification <https://envi.geoscience.cn/help/Subsystems/envi/Content/ExploreImagery/ENVIHeaderFiles.htm> is used.

- byteorder - numeric (integer). Byte order.
- bands - numeric(integer). Number of bands/layers
- nband - numeric(integer). Number of bands/layers
- nlayer - numeric(integer). Number of bands/layers
- layers - numeric(integer). Number of bands/layers
- compress - integer (numeric) or logical. *For create_envi only.* Should ENVI binary file be compressed after closing connection.
- wkt - integer (numeric) or logical. Forced adding 'coordinate system string' to ENVI header file
- ext - character. *For create_envi only.* Extension of ENVI binary file. For extensions not in c("envi", "bin", "dat", "img") list

If file name is unknown, then random file name is used with informing via message().

Value

Object of class `ursaRaster` with opened connection of GDAL or ENVI binary file.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

`ursa_new` creates object of class `ursaRaster` in memory and allows to assign values at once.

Use `session_grid` to check or specify parameters of grid before calling `create_envi`.

Use `[<-` to assign values to ENVI binary file after calling `create_envi`.

Use `close` (or `close_envi`) to close [connections](#).

Examples

```
session_grid(NULL)
a <- create_envi()
fname <- a$con$fname
dir(pattern=basename(envi_list(fname)))
close(a)
invisible(envi_remove(fname))

a <- create_envi("exam1", layername=paste("Band", 1:5)
                , ignorevalue=99, datatype="Int16", interleave="bil")
ursa_info(a)
print(a[])
close(a)
invisible(envi_remove("exam1"))
```

cubehelix	<i>Generate "cubehelix" palette.</i>
-----------	--------------------------------------

Description

cubehelix returns set of RGB colours, which are screen display of intensity images

Usage

```
cubehelix(n, value = numeric(), weak = NA, rich = NA, rotate = NA, hue = NA, gamma = 1,
          dark = NA, light = NA, bright = NA, inv = NA, shuffle = FALSE, verbose = NA)
```

Arguments

n	Positive integer. Length of returned color vector. If n is missing and length of value is positive, then length of value. If missing n and empty value, then n=256.
value	Numeric vector of values, which are associated with a palette. If both positive and negative values are in this vector, then divergence color palette is returned. Default in numeric of length zero (unspecified).
weak	Numeric. The angle (in degrees) of the helix for color with light intensity. If both rich and weak are specified, the rotate is defined as difference between rich and weak. If all weak, rich and rotate are unspecified, then random values are used. Default is NA (unspecified).
rich	Numeric. The angle (in degrees) of the helix for color with dark intensity. If both rich and weak are specified, the rotate is defined as difference between rich and weak. If all weak, rich and rotate are unspecified, then random values are used. Default is NA (unspecified).
rotate	Numeric. The angle of rotation (in degrees) of the helix over the scale; can be negative. If rotate and weak are specified, then rich is defined as sum of weak and rotate. If rotate and rich are specified, then weak is defined as difference between rotate and weak. If all weak, rich and rotate are unspecified, then random values are used. Default is NA (unspecified).

hue	Non-negative numeric. Saturation of color. hue=0 gives pure greyscale. If unspecified, then random value in interval [0.9, 1.5] is used. Default is NA (unspecified).
gamma	Numeric. Power of intensity. Intensity is between dark and light, which are normalized to interval [0, 1]. gamma changes normalized intensity to $\text{intensity}^{\text{gamma}}$. Default is 1.
dark	Positive numeric in interval between 0 and 255. The intensity of the darkest color in the palette. For light backgrounds default is 63. For dark backgrounds default is 14 (inverse order with light).
light	Positive numeric in interval between 0 and 255. The intensity of the lightest color in the palette. For light backgrounds default is 241, for dark backgrounds default is 192 (inverse order with dark).
bright	Positive numeric in interval between 0 and 255. Value for equal intensity for dark and light in the palette. Applied only for both dark=NA and light=NA.
inv	Logical. Inversion of color intensity. If TRUE then color vector is reversed before return. Default is NA, which is interpreted as FALSE.
shuffle	Logical. Should sequence of colors be rearranged occasionally. If TRUE then color vector is rearranged before return. Default is NA, which is interpreted as FALSE.
verbose	Logical. Value TRUE provides information about cube helix on console. Default is NA, which is interpreted as FALSE.

Details

This is modified source code of function `cubeHelix()` from package **rje** under GPL>=2 license.

The palette design is oriented that figures can be printed on white paper. Under this assumption, light color is for small values, and dark color is for big values. In some computer vision and GIS software black background is used, and in this case light color for big values, and dark color of small values looks more naturally. For some thematic maps big values are light, and small values are small (for example, sea ice concentration: open water is blue, close ice is white). RGB and Grayscale remote sensing and photo imagery use light colors for strong signal, and dark colors for weak signal.

Light background is default for figure (specified by argument `background` in function [compose_open](#)).

The palette divergency can be defined only if value is specified. If all values are positive, or all values are negative, then returned palette is not divergent. For divergent palettes the helix sequence is continuous.

If dark and light are unspecified, the color contrast between dark and light drops on reducing number of colors in returned vector.

Value

Vector of RGB color specification.

Acknowledgements

Dave Green, Robin Evans

Author(s)

Dave Green

Robin Evans

Nikita Platonov <platonov@sev-in.ru>

References

Dave Green's 'cubehelix' colour scheme.

Green, D. A., 2011, 'A colour scheme for the display of astronomical intensity images', Bulletin of the Astronomical Society of India, 39, 289. <http://astron-soc.in/bulletin/11June/289392011.pdf> (pre-print at 'arxiv.org')

rje::cubeHelix(); **rje** at CRAN: <https://CRAN.R-project.org/package=rje>

Examples

```
session_grid(NULL)
set.seed(352)
session_grid(regrid(mul=1/16))
a <- ursa_dummy(3,min=0,max=255)
b4 <- b3 <- b2 <- b1 <- vector("list",length(a))
for (i in seq_along(b1)) {
  b1[[i]] <- colorize(a[i],pal=cubehelix(11,weak=45*i,rotate=+270),ncolor=11)
  b2[[i]] <- colorize(a[i],pal=cubehelix(11,weak=45*i,rotate=-270),ncolor=11)
  b3[[i]] <- colorize(a[i]-127,pal=cubehelix)
  hue <- sample(seq(2)-1,1)
  s <- ifelse(hue==0,NA,runif(1,min=91,max=223))
  b4[[i]] <- colorize(a[i]-127,pal=cubehelix,pal.hue=hue,pal.dark=s,pal.light=s)
}
display(c(b1,b2),layout=c(2,NA),decor=FALSE)
display(c(b3,b4),layout=c(2,NA),decor=FALSE)
```

dim

Dimension of multiband raster image

Description

Retrieve the dimension of an object of class `ursaRaster`. The replacement function is `dummy`; it doesn't change raster dimension.

Usage

```
## S3 method for class 'ursaRaster'
dim(x)
## S3 replacement method for class 'ursaRaster'
dim(x) <- value
```


Arguments

x	Object of class <code>ursaRaster</code>
value	Any. Ignored

Details

Use extract operator `[]` and combine function `c` to change third (e.g., temporal) dimension of raster.
 Use `regrid` function to change grid parameters and to resize/resample raster into new grid.

Value

The 'Extract' function `dim` returns named integer vector of length three: 1) number of lines/rows, 2) number of samples/columns, 3) number of bands/channels/layers.
 The 'Replacement' function `dim<-` returns `ursaRaster` object without changes.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
session_grid(regrid(mul=1/16))
a <- ursa_dummy(nband=3)
ursa_info(a)
print(dim(a))
dim(a) <- c(25,00,34)
print(dim(a))
b <- create_envi("tmp1",bandname=letters[1:5],compress=FALSE)
print(dim(b))
close(b)
envi_remove("tmp1")
```

discolor

Destroy color table for raster images.

Description

If raster's categories are integer or numeric, then raster values are restored from names of categories. Otherwise only category names are dropped.

Usage

```
discolor(obj, nodata = NA)
```

Arguments

obj	Object of class <code>ursaRaster</code>
nodata	Numeric. Flag value for "no-data". If NA, then no-data values are missed. Default is NA.

Value

Object of class `ursaRaster` without color table.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- colorize(pixelsize(),ncolor=7)
print(ursa_colortable(a))
print(a)
b <- discolor(a)
print(ursa_colortable(b))
print(b)
```

display

Plot raster image(s) in the PNG format.

Description

High-level function to create multi-panel layout of images and to display with decoration (gridlines, coastlines, scalebar, colorbars) in the PNG format. It is an aggregator of low-level functions for typical plotting.

Usage

```
display(obj, ...)
```

Arguments

obj	Object of class <code>ursaRaster</code> or list of <code>ursaRaster</code> objects.
...	Passed to either display_brick or display_stack or display_rgb functions and further to hierarchy of plotting functions: • compose_open – compose_design • compose_plot – panel_new – panel_raster

- panel_decor
 - * panel_graticule
 - * panel_coastline
 - * panel_scalebar
 - * panel_annotation
- compose_legend
- compose_close

Details

If argument `obj` is missing (e.g, calling `display()` without parameters) then plotting the sessional CRS with blank image.

If argument `obj` is `list` of `ursaRaster` objects (or object of class `ursaStack`) then `display_stack` is called.

If argument `obj` is object of class `ursaRaster` and has 3 or 4 bands and values in each band are `integer` and in interval between 0 and 255, then `display_rgb` is called.

If argument `obj` is object of class `ursaRaster` then firstly internal test is applied to detect either image's bands contains homogeneous information (raster brick) or heterogeneous information (raster stack). Then either `display_brick` or `display_stack` is called. This test is rough due to unknown data origin. It is supposed to adjust kind of plotting by means of direct specification of `display_brick` or `display_stack`.

Value

Returned value from either `display_brick` or `display_stack` or `display_rgb` functions.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

`display_brick`, `display_stack`, `display_rgb`

R-styled plotting: `plot`, `image`

Examples

```
session_grid(NULL)
set.seed(500)
a.brick <- a.stack <- ursa_dummy(nband=3,min=0,max=255,mul=1/16)
a.stack[2] <- a.stack[2]/10
a.stack[3] <- sqrt(a.stack[3])
a.rgb <- as.integer(round(a.brick))
print(a.brick)
print(a.stack)
print(a.rgb)
display(a.brick,decor=FALSE)
display(a.stack,decor=FALSE)
display(a.rgb)
```

display_brick

*Plot multi-band homogenous raster image in the PNG format.***Description**

Raster image is forced to be interpreted as homogenous (having the same units). It implies creating multi-panel layout with multiple colorbars.

Usage

```
display_brick(obj, ...)
display_homo(obj, ...)
```

Arguments

obj	Object of class <code>ursaRaster</code> or <code>list</code> of <code>ursaRaster</code> objects.
...	Passed to hierarchy of plotting functions: • <code>compose_open</code> – <code>compose_design</code> • <code>compose_plot</code> – <code>panel_new</code> – <code>panel_raster</code> – <code>panel_decor</code> * <code>panel_graticule</code> * <code>panel_coastline</code> * <code>panel_scalebar</code> * <code>panel_annotation</code> – <code>compose_legend</code> • <code>compose_close</code>

Details

If argument `obj` is `list` of `ursaRaster` objects (or object of class `ursaStack`) then `obj` is coerced to class `ursaRaster` ('stack' is coerced to 'brick').

`display_homo` is a synonym to `display_brick`. It is introduced to emphasize the plotting of homogenous object.

Value

Function returns NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[display](#), [display_stack](#), [display_rgb](#)

Examples

```
session_grid(NULL)
a <- ursa_dummy(nband=3,min=0,max=250)
a[2] <- -a[1]
a[3] <- sqrt(a[1])
a2 <- ursa_stack(a)
print(a2)
display(a2) # likely 'display_stack' will be called
display_brick(a2,stretch="eq",labels=c(-150,-100,0,10,12,20,100,150))
```

display_rgb

Plot RGB (RGBA) color composition in the PNG format.

Description

Raster images are forced to be interpreted as color composition with 3 (RGB) or 4 (RGBA) channels. Values should be in the range between 0 and 255.

Usage

```
display_rgb(obj, ...)
```

Arguments

obj	Object of class <code>ursaRaster</code> or list of <code>ursaRaster</code> objects.
...	Passed to hierarchy of plotting functions: • compose_open – compose_design • compose_plot – panel_new – panel_raster – panel_decor * panel_graticule * panel_coastline * panel_scalebar • compose_close

Details

If argument `obj` is [list](#) of `ursaRaster` objects (or object of class `ursaStack`) then `obj` is coerced to class `ursaRaster` ('stack' is coerced to 'brick').

Colorbar is not plotted.

By default, the created PNG has 24 bits per pixel. This is equal to parameter `bpp=24` ([compose_close](#)). It is allow to specify other value, e.g., `display_rgb(a, bpp=8)`.

By default, labels of gridlines are located in bottom and left sides of the panel with raster. This is equal to parameter `margin=c(TRUE, TRUE, FALSE, FALSE)` ([panel_graticule](#)). It is allow to specify other value, e.g., `display_rgb(a, margin=T)`.

Currently, for color compositions the argument `useRaster` ([panel_raster](#)) is introduced to fix possible coordinate mismatch for Cairo-devices, but have never used.

Value

Function returns NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[display](#), [display_brick](#), [display_stack](#)

Examples

```
session_grid(NULL)
a <- ursa_dummy(nband=3)
display_rgb(a)
```

display_stack

Plot multi-band heterogenous raster images in the PNG format.

Description

Raster images are forced to be interpreted as heterogenous (having the different units). It implies creating multi-panel layout with multiple colorbars.

Usage

```
display_stack(obj, ...)
display_hetero(obj, ...)
```

Arguments

- obj Object of class `ursaRaster` or `list` of `ursaRaster` objects.
- ... Passed to hierarchy of plotting functions:
- `compose_open`
 - `compose_design`
 - `compose_plot`
 - `compose_panel`
 - * `panel_new`
 - * `panel_raster`
 - * `panel_decor`
 - `panel_graticule`
 - `panel_coastline`
 - `panel_scalebar`
 - * `panel_annotation`
 - `compose_legend`
 - * `legend_colorbar`
 - `compose_close`

Details

If argument `obj` is object of class `ursaRaster` then `obj` is coerced to `list` of `ursaRaster` objects ('brick' is coerced to 'stack').

The plot layout is either two-columns or two-rows. Extent of coordinate grid has a form of rectangle. The layout selection depends on ratio of rectangle's sides. For single-column design use parameter `layout=c(NA,1L)`. e.g., `display_brick(a,layout=c(NA,1))`, for single-row design use parameter `layout=c(1,NA)`. The same is for forcing of two-columns (`layout=c(NA,2L)`) and two-rows layouts (`layout=c(2L,NA)`). Other layouts are not applicable for multiple colorbars.

`display_hetero` is a synonym to `display_stack`. It is introduced to emphasize the plotting of complex object with heterogeneous elements, for example, having different units.

Value

Function returns `NULL` value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

`display`, `display_brick`, `display_rgb`

Examples

```
session_grid(NULL)
a <- ursa_dummy(nband=3)
display_stack(a)
```

Description

Management of ENVI files similar to functions of OS file manager.

Usage

```

envi_exists(pattern = "+", path = ".", all.files = FALSE, full.names = TRUE,
            recursive = FALSE, ignore.case = TRUE, exact = FALSE)
envi_list(pattern = "+", path = ".", all.files = FALSE, full.names = recursive,
          recursive = FALSE, ignore.case = TRUE, exact = FALSE)
envi_remove(pattern = "+", path = ".", all.files = FALSE, full.names = recursive,
            recursive = FALSE, ignore.case = TRUE, verbose = FALSE)
envi_copy(src, dst, overwrite = TRUE)
envi_rename(src, dst, overwrite = TRUE)

ursa_exists(fname)

```

Arguments

pattern	Either filename (like basename function) or mask in format of regular expressions or full path name.
path	Either path name (like dirname function) or ignored if pattern describes full path.
all.files	A logical value. If FALSE, only the names of visible files are returned. If TRUE, all file names will be returned. Similar to all.files argument in list.files function
full.names	A logical value. If TRUE, the directory path is prepended to the file names to give a relative file path. If FALSE, the file names (rather than paths) are returned. Similar to full.names argument in list.files function
recursive	Logical. Should the listing recurse into directories? Similar to recursive argument in list.files function
ignore.case	Logical. Should pattern-matching be case-insensitive? Similar to ignore.case argument in list.files function
exact	Logical. Attempt to cancel regular expressions.
verbose	Logical. TRUE provides some additional information on console.
src	Strings of length 1 or more. Name or directory name or path of source ENVI files.
dst	Strings of length 1 or more. Name or directory name path of destination ENVI files. Length is assuming to be equal to length of src
overwrite	Logical. TRUE overwrites destinations ENVI files. FALSE does nothing if destination ENVI file exists.
fname	Character. Full file name or file pattern with file path.

Details

Functions do not view content of any files. The major identifier of ENVI files in file system is ENVI header (*.hdr) file. Binary file is searching along 1) original *.envi, *.bin, *.img, *.dat extensions, 2) externally packing *.gz, *.bz2, *.xz extensions, or 3) packed by this package *.envigz, *.bingz extensions. Functions `envi_copy()` and `envi_rename()` keeps original extension of ENVI binary file; use `file.rename` to rename ENVI binary file.

Value

`envi_exists()` returns integer number of found ENVI files.

`envi_list()` returns character vector of found ENVI files.

`envi_remove()` returns character vector of deleted ENVI files.

`envi_copy()` returns 0L.

`envi_rename()` returns value of `file.rename`, which is applied to objects in file system.

`ursa_exists()` returns TRUE if any *.tif, *.tiff, *.bin, *.hfa file is found.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
wd <- setwd(tempdir())
a1 <- create_envi("tmp1.envi")
a2 <- create_envi("tmp2.")
close(a1,a2)
envi_list()
envi_copy("tmp1","tmp3")
envi_copy("tmp2","tmp4")
envi_list()
envi_rename("tmp3","tmp5")
envi_list()
envi_exists("nofilewithsuchname")
envi_exists("tmp[34]")
envi_remove(".*")
envi_list()
setwd(wd)
```

Extract

Extract portion of raster images

Description

This operator is used to get single band or subset of bands from multi-band `ursaRaster` object. Another purpose is get portions of data including data reading from files.

Usage

```
## S3 method for class 'ursaRaster'
x[i, j, ..., drop = FALSE]
```

Arguments

x	ursaRaster object
i	Integer or character. If integer, then band index, specifying bands to extract. If character, either list of band names or character sting for regular expression to match band index. In the (<i>spatial</i> , <i>temporal</i>) interpretation of ursaRaster object j points to <i>temporal</i> component.
j	Integer. Line index, specifying lines to extract.
...	Mentioned for consistence with internal generic function [] . Use regexp=FALSE for matching by match , and regexp=TRUE for matching by Perl-compatible regexps case insensitive grep . Default is FALSE.
drop	Not used. For consistence with generic function.

Details

Operator `\Quote{[]}` is high-level implementation for data reading. If `x$value` item is not applicable, then value of ursaRaster is not in memory. In this case the controlled by `i` and `j` portion is read to memory. If both `i` and `j` are missing, then `x[]` reads all values from file.

`x[,j]` is appropriate for time series analysis and processing in the case bands have relation to *temporal* observation. Use [regrid](#) for geographical subset and cropping.

Value

ursaRaster object with extractor bands. Values (`$value` item) are in memory.

Warning

It is not allowed to read simultaneously portion of bands and portion of lines from file, *e.g.*,

```
x <- open_envi(fname)
y <- x[2:3,10:20]
close(x)
```

Such brunch is not implemented in code. You use one of the followed tricks:

```
x <- open_envi(fname)
y <- x[2:3][,20:30]
close(x)
```

or

```
x <- open_envi(fname)
y <- x[,20:30][2:3]
close(x)
```

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```

session_grid(NULL)
## Prepare
session_grid(regrid(mul=1/8))
a <- pixelsize()
w <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday",
      "MondayAgain")
b <- rep(a/mean(a), length(w))+seq(length(w))-1
bandname(b) <- w
nr <- ursa_rows(b)
bottom <- (as.integer(nr/2)):nr
write_envi(b, "tmp1", compress=FALSE, interleave="bil")

## Extract
print(b["Monday", regexp=TRUE])
print(b["Monday", regexp=FALSE])
print(b["s"])
print(b["^s"])
d1 <- b[6, bottom]
rm(b)

## Read from file
b <- open_envi("tmp1")
print(b[])
print(b[-c(6:8)])
d2 <- b[, bottom][6] ## don't use b[6, bottom]
close(b)
envi_remove("tmp1")

## Compare
print(d1)
print(d2)

```

focal_extrem

Extremal spatial filter for image

Description

For each band and for each cell, depending of specification, function finds either minimal or maximal value inside of square window. *Focal* operation of map algebra.

Usage

```

focal_extrem(x, kind = c("min", "max"), size = 3, cover = 1e-06,
             fillNA = FALSE, saveMargin = TRUE, verbose = 0L)

```

```
focal_min(x, size = 3, cover = 1e-06, fillNA = FALSE, saveMargin = TRUE, verbose = 0L)
focal_max(x, size = 3, cover = 1e-06, fillNA = FALSE, saveMargin = TRUE, verbose = 0L)
```

Arguments

<code>x</code>	Object of class <code>ursaRaster</code> .
<code>kind</code>	Character. What kind of extremum is required. Allowed values "min" or "max".
<code>size</code>	Positive numeric. Odd values (3, 5, 7, ...) are allowed, but if other value is specified, then it expanded to the next odd value not less than original value. Default is 3L.
<code>cover</code>	Numeric. $0 \leq \text{cover} \leq 1$. Quota for NA values in the focal window in relation to the squared size of the focal window. Quota exceeding leads to recording NA value in the cell. Default is <code>cover=1e-6</code> .
<code>fillNA</code>	Logical. If TRUE then only NA values of source image can be changed, and non-NA values of source image are kept without changes. It may provide less reducing of spatial resolution in the task of spatial interpolation. Default is FALSE.
<code>saveMargin</code>	Logical. If TRUE then adaptive window size is used for cells, where original window goes over image boundary. If FALSE then image is expanded to the half size of focal window by NA values and argument <code>cover</code> is applied to this expanded part. Default is TRUE.
<code>verbose</code>	Integer of 0L, 1L, or 2L, or logical, which is coerced to integer. The level of verbosity. Values >0 provide some additional information on console, <code>verbose=1L</code> is less detailed, <code>verbose=2L</code> is more detailed. Default is 0L.

Details

`focal_min(x, ...)` is a wrapper to `focal_extrem(x, "min", ...)`
`focal_max(x, ...)` is a wrapper to `focal_extrem(x, "max", ...)`

Value

Object of class `ursaRaster` with the same number of bands as in input raster.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

Other focal operations: [focal_mean](#), [focal_median](#).

Examples

```
session_grid(NULL)
a <- ursa_dummy(nband=2,mul=1/8,elements=32)
a[a<80] <- NA
b.min <- focal_extrem(a,"min",size=4,cover=0.5,verbose=1L)
b.max <- focal_extrem(a,"max",size=4,cover=0.5,verbose=1L)
print(list(src=a,min=b.min,max=b.max,dif=b.max-b.min))
```

focal_mean	<i>Low-pass spatial filter for image.</i>
------------	---

Description

Low-pass filtering by a square window in the image processing. *Focal* operation of map algebra. Weight of pixels is proportional to cell area inside of focal window.

Usage

```
focal_mean(x, size = 3.0, cover = 1e-06, fillNA = FALSE, saveMargin = TRUE,
           ,noNA = TRUE, verbose = 0L)
```

Arguments

x	Object of class <code>ursaRaster</code> .
size	Positive numeric. Size of square focal window. Fractional values are allowed. If size is not odd (3, 5, 7, ...), then window size is expanded to the nearest odd value not less than original value, and pixels on border are taken with the weight, which is proportional to the cell area inside of original size. Default size=3.
cover	Numeric. $0 \leq \text{cover} \leq 1$. Quota for NA values in the focal window in relation to all values. Values are taken with the weight proportional of cell areas inside of focal window. Quota exceeding leads to recording NA value in the cell. Default is <code>cover=1e-6</code> .
fillNA	Logical. If TRUE then only NA values of source image can be changed, and non-NA values of source image are kept without changes. It may provide less reducing of spatial resolution in the task of spatial interpolation. Default is FALSE.
saveMargin	Logical. If TRUE then adaptive window size is used for cells, where original window goes over image boundary. If FALSE then image is expanded to the half size of focal window by NA values and argument <code>cover</code> is applied to this expanded part. Default is TRUE.
noNA	Logical. If TRUE then NA values are transformed to numerical constant, which is interpreted as "no data" value. Filter without NA values has more performance, and generally filter with pre- and post-transformations of NA values have more performance too. Default is TRUE.
verbose	Integer of 0L, 1L, or 2L, or logical, which is coerced to integer. The level of verbosity. Values >0 provide some additional information on console, <code>verbose=1L</code> is less detailed, <code>verbose=2L</code> is more detailed. Default is 0L.

Details

The reference is always central pixel, even if window size is even.

If size=3 then multiplier is 3^{-2} and elements have equal weights:

```

      [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
[3,]    1    1    1

```

If size=2 then multiplier is 2^{-2} and weights of elements are:

```

      [,1] [,2] [,3]
[1,] 0.25 0.50 0.25
[2,] 0.50 1.00 0.50
[3,] 0.25 0.50 0.25

```

If size=3.4 then multiplier is 3.4^{-2} and weights of elements are:

```

      [,1] [,2] [,3] [,4] [,5]
[1,] 0.04 0.20 0.20 0.20 0.04
[2,] 0.20 1.00 1.00 1.00 0.20
[3,] 0.20 1.00 1.00 1.00 0.20
[4,] 0.20 1.00 1.00 1.00 0.20
[5,] 0.04 0.20 0.20 0.20 0.04

```

Value

Object of class `ursaRaster` with the same number of bands as in input raster.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```

session_grid(NULL)
a <- ursa_dummy(nband=1,mul=1/8,elements=0)
a[a<80] <- NA
print(a)
b1 <- focal_mean(a,size=6,cover=0.5,saveMargin=FALSE)
b2 <- focal_mean(a,size=6,cover=0.5,saveMargin=TRUE)
b3 <- focal_mean(a,size=6,cover=0.5,saveMargin=TRUE,fillNA=TRUE)
print(b3-a)
display(c(a,b1,b2,b3),blank.angle=c(-45,45),blank.density=20)

```

focal_median

Median spatial filter for image

Description

For each band and for each cell, function finds median value inside of square window. *Focal* operation of map algebra.

Usage

```
focal_median(x, size = 3, cover = 1e-06, fillNA = FALSE, saveMargin = TRUE, verbose = 0L)
```

Arguments

<code>x</code>	Object of class <code>ursaRaster</code> .
<code>size</code>	Positive numeric. Odd values (3, 5, 7, ...) are allowed, but if other value is specified, then it expanded to the next odd value not less than original value. Default is 3L.
<code>cover</code>	Numeric. $0 \leq \text{cover} \leq 1$. Quota for NA values in the focal window in relation to the squared size of the focal window. Quota exceeding leads to recording NA value in the cell. Default is <code>cover=1e-6</code> .
<code>fillNA</code>	Logical. If TRUE then only NA values of source image can be changed, and non-NA values of source image are kept without changes. It may provide less reducing of spatial resolution in the task of spatial interpolation. Default is FALSE.
<code>saveMargin</code>	Logical. If TRUE then adaptive window size is used for cells, where original window goes over image boundary. If FALSE then image is expanded to the half size of focal window by NA values and argument <code>cover</code> is applied to this expanded part. Default is TRUE.
<code>verbose</code>	Integer of 0L, 1L, or 2L, or logical, which is coerced to integer. The level of verbosity. Values >0 provide some additional information on console, <code>verbose=1L</code> is less detailed, <code>verbose=2L</code> is more detailed. Default is 0L.

Value

Object of class `ursaRaster` with the same number of bands as in input raster.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

Other focal operations: [focal_mean](#), [focal_extrem](#).

Examples

```
session_grid(NULL)
a <- ursa_dummy(1,mul=1/8,elements=0,bandname="src")
a[a<80] <- NA
bF <- c(fillNA.F=focal_median(a[1],size=5,cover=0.5,fillNA=FALSE))
bT <- c(fillNA.T=focal_median(a[1],size=5,cover=0.5,fillNA=TRUE))
print(c(diff=bT-bF))
d <- c(a,bF,bT)
print(d)
display(d)
```

focal_special	<i>Custom spatial filtering for image</i>
---------------	---

Description

For each band and for each cell, function calculates value using specific matrix of square window. *Focal* operation of map algebra.

Usage

```
focal_special(x, type = c("custom", "gaussian", "laplacian", "osisaf",
                          "hires", "correl", "LoG", "sobel", "sobelG"),
              fmask = NULL, size = 3, alpha = 0.5, sigma = (size-1)/4,
              cover = 1 - 1e-06, fillNA = FALSE, saveMargin = FALSE, verbose = 0L)
```

Arguments

x	Object of class <code>ursaRaster</code> .
type	Character, which is checked by match.arg .
fmask	Numeric square matrix. Filter mask. If NULL then <code>matrix(1, ncol=1)</code> is used. Default is NULL.
size	Numeric. Diameter of circuled filter mask. Coerced to the nearest odd value not less than original value.
alpha	Nimeric. Parameter alpha for "Laplacian", "LoG", "hires", "correl" filters. Ignored for others. Default is 0.5.
sigma	Numeric. Parameter sigma for "Gaussian", "LoG" filters. Ignored for others. Default is 0.5.
cover	Numeric. $0 \leq \text{cover} \leq 1$. Quota for NA values in the focal window in relation to the squared size of the focal window. Quota exceeding leads to recording NA value in the cell. Default is <code>cover=1-1e-6</code> .
fillNA	Logical. If TRUE then only NA values of source image can be changed, and non-NA values of source image are kept without changes. It may provide less reducing of spatial resolution in the task of spatial interpolation. Default is FALSE.
saveMargin	Logical. If TRUE then adaptive window size is used for cells, where original window goes over image boundary. If FALSE then image is expanded to the half size of focal window by NA values and argument <code>cover</code> is applied to this expanded part. Default is FALSE.
verbose	Integer of 0L, 1L, or 2L, or logical, which is coerced to integer. The level of verbosity. Values >0 provide some additional information on console, <code>verbose=1L</code> is less detailed, <code>verbose=2L</code> is more detailed. Default is 0L.

Details

Developed under impression from Matlab's "fspecial".

- type="custom"
Filter mask (argument fmask) should be specified manually.
- type="gaussian"
Gaussian filter. For cascade filtering (sequence of increasing or decreasing window size) $\sigma = (\text{size}-1)/4$ produces the same distribution density relative to window size. If σ is high but not Inf then it is low-pass filter with diameter=size of circular focal window.
- type="laplacian"
Laplacian filter. Only size=3 makes sense. Any size is coerced to size=3.
- type="osisaf"
Filter for edge detection. Only size=5 makes sense. Any size is coerced to size=5. *TODO (pl): reference*

```
-0.0625 -0.0625 -0.0625 -0.0625 -0.0625
-0.0625  0.1250  0.1250  0.1250 -0.0625
-0.0625  0.1250  0.0000  0.1250 -0.0625
-0.0625  0.1250  0.1250  0.1250 -0.0625
-0.0625 -0.0625 -0.0625 -0.0625 -0.0625
```

- type="hires"
Filter for unsharpening. Only size=3 makes sense. Any size is coerced to size=3.
- ```
-alpha alpha-1 -alpha
alpha-1 alpha+5 alpha-1
-alpha alpha-1 -alpha
```
- type="correl"  
Filter for contrast increasing. Only size=3 makes sense. Any size is coerced to size=3.
- ```
alpha^2      -alpha*(1+alpha^2)      alpha^2
-alpha*(1+alpha^2)      (1+alpha^2)^2      -alpha*(1+alpha^2)
alpha^2      -alpha*(1+alpha^2)      alpha^2
```
- type="LoG"
Laplacian of Gaussian. Filter for edge detection. σ is used, α is ignored.
 - type="sobel"
Two-directional Sobel filtering. Only size=3 makes sense. Any size is coerced to size=3.
 - type="sobelG"
Sobel gradient. Only size=3 makes sense. Any size is coerced to size=3.

Value

Object of class `ursaRaster` with the same number of bands as in input raster.

Warning

Laplacian of Gaussian filter (type="LoG") is not implemented clearly due to applying continuous-valued formula to discrete matrix.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

References

TODO(pl): at least reference to 'osisaf'.

See Also

Other focal operations: [focal_mean](#), [focal_median](#), [focal_extrem](#).

Examples

```
session_grid(NULL)
v <- round(runif(8,min=-1,max=1),3)
customFilter <- matrix(c(v[1:4],-sum(v),v[5:8]),ncol=3)
a <- ursa_dummy(1,mul=4/8,elements=32)
tpList <- eval(formals("focal_special")$type)
res <- c(src=a,as.ursa(bandname=tpList))
for (tp in tpList) {
  message(tp)
  res[tp] <- focal_special(a,tp,fmask=customFilter,size=11,sigma=1,alpha=0.8
    ,saveMargin=0,verbose=2L)
}
print(res)
display(res,decor=FALSE)
```

get_earthdata	<i>Retrive data from Global Imagery Browse Services (GIBS) using API for Developers</i>
---------------	---

Description

get_earthdata allows retrieving georeferences optical satellite images of low and moderate spatial resolution (up to 250m per cell) using GIBS API for Developers.

Usage

```
get_earthdata(bbox = NA, res = c("2km", "1km", "500m", "250m"),
  date = NA, product = "", geocode = "",
  expand = 1.05, border = 0, display = FALSE,
  cache = NA, verbose = FALSE)
```

Arguments

bbox	Numeric of length 4 or character. Spatial extent in the notation <code>c(minx,miny,maxx,maxy)</code> . Can be in units of meters or degrees. If all absolute values are less than 360, then units are in degrees and projection is EPSG:3857, else units are in meters and projection is EPSG:3413. If <code>bbox=NULL</code> , then function return list of available products. If <code>bbox=NA</code> then boundary box is attempted for taking from session grid. If character, then boundary box is taken from geocoding. Default is region of Vaigach Island.
res	Character or numeric. Parameter, which is responsible for dimension of output raster image. If character, then zoom is selected using keyword list <code>c("2km", "1km", "500m", "250m")</code> for EPSG:3413. If <code>res<10</code> then it is interpreted as zoom for Web Map Tile Service (WMTS). If <code>res>100</code> then <code>res</code> is interpreted as preferred image dimension. If <code>res=NA</code> then <code>res=480L</code> .
date	Character or "Date" object. Date for image retrieving. Default is <code>Sys.Date()-1L</code> .
product	Character of integer. Data product form GIBS. Currently only MODIS-oriented (corrected reflectance) products are available: 1. "MODIS_Aqua_CorrectedReflectance_Bands721" 2. "MODIS_Terra_CorrectedReflectance_Bands721" 3. "MODIS_Aqua_CorrectedReflectance_TrueColor" 4. "MODIS_Terra_CorrectedReflectance_TrueColor" 5. "VIIRS_SNPP_CorrectedReflectance_TrueColor" 6. "Coastlines" Please check actual list by calling <code>get_earthdata(bbox=NULL)</code>. If numeric, then index of item among available products. Regular expressions can be used to simplify value of product, e.g., case-insensitive "aqua 721", "terra truecolor", "suomi", "SNNP".
geocode	Character. Keyword for geocode service. Valid values are "google", "nominatim". Default is ""; several services are considered in the case of failure.
expand	Numeric. Multiplier for plotting panel zoom in relation to extent of plotting geometry. Ignored if geocoding is not applied. Default is 1.05.
border	Integer. Value in pixels of fixed margins around plotting geometry. Ignored if geocoding is not applied. Default is 0L.
display	Logical. Value TRUE forces to display image instead of return it. Default is FALSE.
cache	Logical. Is cache used? Default is NA, which is interpreted as TRUE for any requested date excepting not late time of today (approximately 17:00 UTC).
verbose	Logical. Value TRUE may provide some additional information on console. Default is FALSE.

Details

Argument `method="libcurl"` is used in function [download.file](#) for tile downloading. Please check [capabilities\("libcurl"\)](#).

Valid zoom values (e. g., specified via `res` argument) are 3:6 for EPSG:3413 and 0:8 for EPSG:3587.

Longitude 180 degrees has a seam in EPSG:3857 (e.g., see `bbox=c(170,68,-170,73)` and `bbox=c(-1600000,1308000,-1308000,1308000)` for Wrangel Island. If region crosses longitude 180 degrees in EPSG:3857, then the prior day is taken for Western Hemisphere.

Value

If `bbox=NULL`, then character vector of available products.

If `display=FALSE` then object of class `ursaRaster` with RGBA image.

If `display=TRUE` then returned value of `display_brick`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

References

[GIBS API for Developers](#)

Examples

```
session_grid(NULL)
pr <- get_earthdata()
print(pr,quote=FALSE)

## internet connection is required -- begin
a1 <- get_earthdata(bbox=c(2000000,400000,2300000,700000))
display(a1)

## internet connection is required -- end
a2 <- get_earthdata(product=2
  # ,date=Sys.Date()-7L ## empty tiles for winter polar regions
  ,date=as.Date(format(Sys.Date()-365L,"%Y-06-15"))
  ,res=7,bbox=c(57.8,69.4,62.3,70.8))
display(a2)
```

glance

Command line utility for spatial view of raster or vector GIS file.

Description

`glance` is a parser of command line arguments for non-public function `.glance`, which creates multi-panel plots for each attribute of vector file or for each band of raster file.

Usage

```
glance(...)

# non-public
.glance(dsn, layer = ".*", grid = NULL, field = ".*", size = NA, expand = 1,
        border = 27, lat0 = NA, lon0 = NA, resetProj = FALSE, resetGrid = FALSE,
        style = "auto", feature = c("auto", "field", "geometry"), alpha = NA,
        basemap.order = c("after", "before"), basemap.alpha = NA,
        engine = c("native", "sp", "sf"), geocode = "", place="",
        area = c("bounding", "point", "shape"), zoom = NA, gdal_rasterize = FALSE,
        silent = FALSE, verbose = FALSE, ...)
```

Arguments

dsn	Character or object of either <code>ursaRaster</code> , <code>Spatial</code> , or <code>sf</code> classes. If character, then data source name (interpretation varies by driver - for some drivers, dsn is a file name, but may also be a folder, or contain the name and access credentials of a database).
layer	Character or integer. If integer, then layer index. If character, then pattern (regular expressions) to recognize layer by name. Only one layer selection is allowed. If selected more than one layer, the error message contains indices and names of layers. Usually, datasets (e. g., "ESRI Shapefile") have only one layer. Default is <code>".*"</code> ; interpreted as all layers.
grid	Object of class <code>ursaGrid</code> or <code>NULL</code> . Reference CRS and boundary box for visualization. If <code>NULL</code> , then CRS and boundary box are zoomed to layer. Default is <code>NULL</code> .
field	Character. Pattern for field (attribute, column,...) selection by name using regular expressions . Multiple selection is allowed. Default is <code>".*"</code> ; all fields.
size	Integer of length 1 or 2 or character of length 1. Size of plotting panel in pixels. If character, then parsed to integer of length 1 or 2. Length 2 is used only for web cartography. If length 1, then size defines width of panel, and height is defined automatically. If integer, then width of panel for plotting in pixels. Default is <code>NA</code> ; for web cartography value of maximal size of static maps, and 640 for other cases.
expand	Numeric. Multiplier for plotting panel zoom in relation to extent of plotting geometry. Default is <code>1.0</code> .
border	Integer. Value in pixels of fixed margins around plotting geometry. Default is 27.
lat0	Numeric. Parallel or zero distortion. If <code>NA</code> , then parallel or zero distortion is determined from object geometry. Actual for <code>"proj=stere"</code> projections. Default is <code>NA</code> .
lon0	Numeric. Central meridian, which have vertical direction on the plot. If <code>NA</code> , then central meridian is determined from object geometry. Default is <code>NA</code> .
resetProj	Logical. Value <code>TRUE</code> overwrites projection of vector file. Default is <code>FALSE</code> .

resetGrid	Logical. If TRUE, then <code>session grid</code> is ignored, and new session grid is assigned from input file. If FALSE, then input file is nested in the session grid.
style	Character. Either projection class or source of web-cartography for basemap. Specified by a sentence of words separated by spaces. • Projection class Valid values are "stere", "laea", "merc", "longlat". Default is keyword "auto"; use object projection, if this projection differs from projection class "+longlat", otherwise, projection ("stere" or "merc") is determined internally. • Web cartography. – Static map Valid values are "google", "openstreetmap", "sputnikmap". Static maps have priority over tile services. however additional word "static" can be specified in the sentence, e.g., "openstreetmap static" or "static google". Additional parameters for request to web-script can be added in the sentence in the form "argument1=value1 [argument2=value2]", e.g., style="google static maptype=terrain language=ru-RU scale=2". – Tile service Supported tile services can be returned by calling of non-public function <code>ursa:::tileService()</code> without arguments. Valid values are "mapnik", "cycle", "transport", "mapsurfer", "sputnik", "thunderforest", "carto", "kosmosnimki", etc. By default, if data has no data fields (e. g., geometry only), then basemap is drawn in color, else in grayscale. Adding word "color" (or "colour") to the sentence forces to use colored basemap. Adding word "gray" (or "grey", "greyscale", "grayscale") to the sentence forces to use colored basemap. The order of words in the sentence is any. Keywords "google", "openstreetmap" force to use "Google Static Map" or "OpenStreetMap static map images" for basemap; the resulted projection has class "+proj=merc".
feature	Character. Appearance of visualization. If "field" then data of each field is plotted on separate panel (number of panels is equal to number of columns in attribute table). If "geometry" then each feature is plotted on separate panel (number of panels is equal to number of rows in attribute table). Default is "auto"; if intersects of features are found, then "geometry" is used, else "field".
basemap.order	Character. The order of basemap layer rendering in the case of web-cartography basemap. If "before", then basemap is plotted before object plot. If "after", then basemap is plotted over object.
basemap.alpha	Character. The saturation of basemap in the case of web-cartography basemap. Default is NA; <code>basemap.alpha=0.5</code> for <code>basemap.order="before"</code> and <code>basemap.alpha=0.35</code> for <code>basemap.order="after"</code> .
alpha	Character. The opacity of plotted object. Default is NA; 0.75 for <code>basemap.order="before"</code> in web-cartography style, 1.00 - in all other cases.
engine	Character keyword. Forcing to vector files processing by functions from package sp (engine="sp") or package sf (engine="sf", if sf is installed). Default is "native"; if dsn is Spatial object or if sf is not installed, then "sp" is used.

geocode	Character. Keyword for geocode service. Valid values are "google", "nominatim". If dsn is character and file dsn not found, then trying to interpret dsn as a request to geocode service. The output is only basemap of web cartography. Default is ""; several services are considered in the case of failure. Is style is not specified, then "Google Static Map" is used for geocode="google", and "OpenStreetMap static map images" for geocode="nominatim".
place	Character. Type of geographical object (river, island) in the geocoding request. If geocode service is "nominatim", then place is searched among attributes "class" and "type". Default is ""; any object is acceptable.
area	Character. Keyword of spatial class of geocoded object. "bounding" is used for boundary box; "point" is used for point. Default value is extracted by <code>match.arg(area)</code> .
zoom	Positive integer or character. Zooming if web-cartography is applied for basemap. If integer, then value of zoom for tile services and staticmap. If character, then "0" means zoom by default (defined internally), "+1" means increment on 1 of default zoom, "+2" means zoom increment on 2, "-1" means zoom decrement on 1, "-2" means zoom decrement on 2, <i>etc.</i> Default is NA; zoom is defined internally.
gdal_rasterize	Logical. If TRUE and GDAL utilities are in the system search path, then overlay for panels is formed via rasterization of vector file. GDAL utility "gdal_rasterize" is used. Note, that GDAL (system level) is optional for this package. Default is FALSE.
silent	Logical. Value TRUE cancels progress bar. Default is FALSE.
verbose	Logical. Logical. Value TRUE may provide some additional information on console. Default is FALSE.
...	glance: Arguments, which are passed to <code>.glance</code> or to <code>display</code> . <code>.glance</code> : Arguments, which are passed to static maps API, <code>colorize</code> , <code>display</code> , <i>etc.</i>

Details

Command line usage implies set of arguments using pair: argument name and argument value. and values in the format "[name1]=value1 [name2]=value2". No spaces around = (equal symbol). Argument name can be omitted, symbol = is omitted too. If argument value has spaces, then argument value should be surrounded by double quotes (`fname="my test.shp"`). If argument value is matched to R function, then such value should be surrounded by single quotes (`layer='density'`).

Command line usage example: `Rscript -e ursa::glance() 'final_more_than_032.sqlite' attr="select" resetProj=TRUE expand=1.5`

For OS Windows, bat-file can be created for raster and vector file association: `Rscript -e ursa::glance() %*`

Command line usage implies external software for PNG view `session_pngviewer(TRUE)`.

Value

glance returns integer: 0L - successful, 10L - call without arguments.

Note

Package **sp** is 'Suggested' for package **ursa**.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
f <- system.file("shape/nc.shp",package="sf")
glance(f,style="merc",field="(NAME|AREA|COUNT)")
cmd <- paste("Rscript --vanilla -e ursa::glance()",paste0("\"",f,"\""),
            ,"style=\"merc\"",field="74$")
cat(" ----- Try in command line: -----\\n")
message(cmd)
cat(" ----- end of quoting -----\\n")

## windows: figure will be opened using *.png file association
try(system(cmd,wait=FALSE))

a <- data.frame(lat=c(70.734,71.657),lon=c(178.577,-177.38),place="Wrangel Island")
if (requireNamespace("sp")) {
  sp::coordinates(a) <- ~lon+lat
  sp::proj4string(a) <- "EPSG:4326"
} else {
  a <- sf::st_as_sf(a,coords=c("lon","lat"),crs=4326)
}

## internet connection is required
glance(a,style="Esri.Satellite",border=0)

## internet connection is required
glance(a,style="opentopomap grey",border=0)

## internet connection is required
glance("Svalbard 9170",resetGrid=TRUE)
```

global operator

Extract certain statistics for whole image

Description

Function from this global.FUN list returns required statistics *FUN* for the whole image.

Usage

```

global_mean(x, ursa = FALSE, ...)
global_median(x, ursa = FALSE, ...)
global_sd(x, ursa = FALSE, ...)
global_sum(x, ursa = FALSE, ...)
global_min(x, ursa = FALSE, ...)
global_max(x, ursa = FALSE, ...)
global_n(x, ursa = FALSE, ...)
global_nNA(x, ursa = FALSE, ...)
global_range(x, ursa = FALSE, ...)
global_quantile(x, ursa = FALSE, ...)

```

Arguments

x	Object of class <code>ursaRaster</code> .
ursa	Logical. The class of returned value. If FALSE then numeric vector of length one is returned (for <code>global_range</code> vector has length two). If TRUE then returned value is single-band raster image (two-bands image for <code>global_range</code>) with constant value for all cells (blank image). Default is FALSE.
...	Arguments in function <code>global.FUN</code> which are passed to function <code>FUN</code> .

Details

For any function `global.FUN`, if argument `na.rm` is not in `...`, then `FUN` is called with forced `na.rm=TRUE`.

`global_range\emph{list of arguments}` is implemented as `c(global_min(list of arguments), global_max(list of arguments))` with the same list of arguments.

Alternative method to get global statistics is function applying directly to the raster value. For example, `sd(ursa_value(x, na.rm=TRUE))`. This way is also appropriate for missing global functions: for example, `var(ursa_value(x, na.rm=TRUE))`.

Value

If `ursa=FALSE` then [numeric](#).

If `ursa=TRUE` then object of class [ursaRaster](#).

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```

session_grid(NULL)
a <- ursa_dummy(2, min=-40, max=80)
a[a<0] <- NA
print(a)
a.mean <- global_mean(a)
a.sd <- global_sd(a)
a.sum <- global_sum(a)

```

```

a.min <- global_min(a)
a.max <- global_max(a)
a.median <- global_median(a)
print(c(mean=a.mean, sd=a.sd, sum=a.sum, min=a.min, max=a.max, median=a.median))
v.max <- max(ursa_value(a), na.rm=TRUE)
print(c('global_max()'=a.max, 'max(ursa_value())'=v.max, dif=a.max-v.max))
r.max <- global_max(a, ursa=TRUE)
print(r.max)
b <- c(a, 'appended scalar value'=a.max)
print(b)
print(global_quantile(a))

```

groupGeneric

Group Generic Functions for raster image

Description

These functions implement arithmetical and logical operations, mathematical functions for objects of class `ursaRaster` as well as group generic functions from package **base** do similar for S3 class. These are *local* operations in the raster algebra (map algebra).

Usage

```

## S3 method for class 'ursaRaster'
Ops(e1, e2 = NULL)

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

## S3 method for class 'ursaRaster'
Complex(z)

## S3 method for class 'ursaRaster'
Summary(..., na.rm = FALSE)

```

Arguments

<code>x</code>	<code>ursaRaster</code> object
<code>e1</code>	<code>ursaRaster</code> object
<code>e2</code>	Numeric of length 1, matrix, array, or <code>ursaRaster</code> object.
<code>na.rm</code>	Logical. If <code>na.rm=TRUE</code> then no-data values are omitted.
<code>z</code>	Any.
<code>...</code>	For group <i>Math</i> - further arguments passed to methods. See description for generic. For group <i>Summary</i> - set of arguments, which are recognized via their names (using regular expressions), position and classes.

.*** Position 1. Object of class `ursaRaster`.

`cov|cvr` Position >1. Numeric between 0 and 1. If proportion of bands with no data for given location exceeds `cover` then output value is NA (no data). Default is $0.5 \cdot 10^{-3}$.

`w` Position >1. Numeric of length number of bands or NULL. Band weights for weighted mean. Default is NULL; all bands have equal weights.

`name` Position >1. Character of length 1. Band name for output raster. Default is ""; band name is assigned automatically.

`verb(ose)*` Position >1. Logical. `verbose=TRUE` provides some additional information on console. Default is FALSE.

Details

The groups are 'Summary', 'Ops', 'Math', and 'Complex'. See “Details” section in the [S3 Generic Functions](#) help page.

The group 'Complex' is unsupported.

The groups 'Math' and 'Summary' are implemented completely.

The group 'Ops' has some features.

- Logical operators "`<`", "`>`", "`<=`", "`>=`", "`==`", "`!=`" return 'NA' for value FALSE and '1' for value TRUE to organize cells' masking.
- Unary operator "`!`" is equal to binary operator operators "`!=`", where the second argument is scalar value 0 (zero).

The operators of groups 'Math' and 'Ops' destroy [color tables](#).

For group 'Summary' the realization of local operators of map algebra is possible via [apply](#) function:

```
apply(ursa_value(obj), 1, function(x) {y <- sd(x)+1;y})
or
as.ursa(apply(obj, 1:2, function(x) {y <- sd(x)+1;y}))
```

Value

Operators of groups 'Complex' return [stop](#)

Operators of groups 'Math', 'Ops', 'Summary' return object of class `ursaRaster`

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

Other S3 generic function for local operations of map algebra are [mean](#), [median](#).

Standard deviation (local) and certain local operations can be extracted using [local_stat](#).

Examples

```

session_grid(NULL)
session_grid(regrid(mul=1/4))
a1 <- ursa_dummy(nband=3,min=-5*pi,max=5*pi)
print(a1)

try(print(complex1 <- Re(a1)))

print(math1 <- a2 <- round(a1))
print(math1 <- sin(a1))
print(math2 <- floor(a1))
print(math3 <- ceiling(a1))
print(math4 <- cumsum(a1)) ## does this have a sense for rasters?

print(ops1 <- a1-2*rev(a1)+mean(a1))
print(mean(ops1)) ## vanishing
a2 <- ursa_new(value=c(1,2,4),bandname=c("single","double","quadruple"))
print(a2)
print(ops2 <- a2[1]==a2[2])
print(ops3 <- a2[1]==a2[2]/2)
print(ops4 <- a1>0)
print(a1[a1>0])

print(sum1 <- sum(a1))
print(sum2 <- range(a1))

```

head

Extract first and last bands of raster image

Description

Functions to extract first bands (head), last bands (tail) and first+last bands (series) of raster image.

Usage

```

## S3 method for class 'ursaRaster'
head(x, n = 3L, ...)

## S3 method for class 'ursaRaster'
tail(x, n = 3L, ...)

series(x, n = 3L, s=170, ...)

```

Arguments

x	Object of class ursaRaster
n	Positive integer. Number of extracted bands.

s	Positive numeric. Maximal size of memory in MB for extracted raster image in the assumption that class of values is numeric .
...	Not used.

Details

Function `series` combines consecutive calling `head(x)`; `tail(x)` with checking the size of extracted part of raster image. If size exceeds specified value of the argument `s`, then number of extracted bands `n` is decreased.

Value

Object of class `ursaRaster`

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
session_grid(regrid(mul=1/8))
a <- ursa_dummy(nband=101)
print(head(a))
print(tail(a))
print(series(a,2))
print(series(a[1:5]))
```

hist

Histogram of raster image

Description

Two functions for manipulation with histograms. In function `hist` values of `ursaRaster` objects are passed to generic function [hist](#), which allows compute and optionally plot histograms. Other function, `histogram`, plots histogram in the graphical device [png](#) directly.

Usage

```
## S3 method for class 'ursaRaster'
hist(x, ...)

ursa_hist(obj, width = 800, height = 600, ...)
histogram(...)
```

Arguments

<code>obj, x</code>	Object of class <code>ursaRaster</code>
<code>width</code>	Positive integer. Width of histogram's panel.
<code>height</code>	Positive integer. Height of histogram's panel.
<code>...</code>	Other arguments, which are passed to <code>colorize</code> and <code>compose_open</code> functions.

Details

`histogram` is synonym of `ursa_hist`.

Function `hist` for `ursaRaster` object is defined as `hist(ursa_value(obj), ...)`.

In the function `histogram` each bin corresponds to category. The image splitting to categories is realized via `colorize` function. The panel of plotting is constructed using artificial coordinate system without geographical projection. The purpose of `compose_open` function is prepare layout for plotting raster images; in the case of histogram, the purpose of this function is prepare layout for plotting histogram

Value

Function `histogram` returns `0L`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

`colorize` is used to define histogram bins.
`compose_open` prepares panel for histogram plotting.
`hist` computes and plots histograms.

Examples

```
session_grid(NULL)
a <- pixelsize()
hist(a)
histogram(a,breaks=21)
```

identify

Get value and coordinates from location

Description

Functions to extract values of raster image from given location, specified by coordinates in raster projection, by cell position or by geographical coordinates. Additional utils to convert cell position and planar coordinates mutually.

Usage

```
value_xy(obj, ...)

value_ll(obj, ...)

value_cr(obj, ...)

coord_xy(obj, ...)

coord_cr(obj, ...)
```

Arguments

- obj Object of class `ursaRaster`.
- ... the set of arguments, which are recognized via their names (using [regular expressions](#)) and classes:

<i>Matched pattern</i>	<i>Function</i>	<i>Used name</i>	
ind	*_*	ind	Index (positive integer) in internal value storage.
^c	*_cr	col	Integer of non-zero length. Index of column/sample Length of column and row
^r	*_cr	row	Integer of non-zero length. Index of row/line. Length of column and row indice
^x	*_xy	x	Numeric of non-zero length. X-axis coordinate in grid of obj. The length of X-
^y	*_xy	y	Numeric of non-zero length. Y-axis coordinate in grid of obj. The length of X-
^lon	value_ll	lon	Longitude. The length of longitudes and latitudes should be the same for creatin
^lat	value_ll	lat	Latitude. The length of longitudes and latitudes should be the same for creating

Details

`value_xy` returns values for location, which is specified by planar coordinates (x, y).
`value_cr` returns values for location, which is specified by cell position (column, row) relative to upper-left corner of image .
`value_ll` returns values for location, which is specified by longitude and latitude (long, lat).

`coord_xy` transforms planar coordinates (x, y) to cell position (column, row).
`coord_cr` transforms cell position (column, row) to planar coordinates (x, y).

It is required to use a couple of coordinate vectors: (x, y), (c, r) or (lon, lat) of the same length. The unary argument is interpreted as index in internal value storage.

Position in column/row coordinates starts from upper-lever corner. The cell of upper-level corner has (1, 1) coordinates (in R indices starts from 1L), whereas in some GIS the same corner cell has (0, 0) coordinates.

The column names of returned matrix are character format of index in internal value storage. This index can be specify in any function as argument `ind` instead of coordinates (planar, geographical, cell position).

Value

For `value.*` numeric matrix of raster values. Band values for specific coordinates are by column. Set of specific coordinates are by row. `rownames` are band names, and `colnames` are index in internal value storage.

For `coord.*` numeric matrix of coordinates with a vector of couple coordinates, one coordinate per one row. `rownames` are returned coordinates, and `colnames` are index in internal value storage.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
set.seed(352)
a <- as.integer(ursa_dummy(3,min=0,max=999))
ind <- which(ursa_value(a[1])==890)
print(ind)
msk <- a[1]==890
am <- a[msk]
b <- as.data.frame(am)
b$jx <- b$x+runif(nrow(b),min=-1000,max=1000)
b$ jy <- b$y+runif(nrow(b),min=-1000,max=1000)
print(b)
cr1 <- coord_xy(a,x=b$jx,y=b$jy)
cr2 <- coord_xy(a,y=b$jy,x=b$jx)
cr3 <- coord_xy(a,ind=ind)
print(cr1)
print(list('cr1 and cr2'=all.equal(cr1,cr2)
          , 'cr2 and cr3'=all.equal(cr2,cr3)
          , 'cr3 and cr1'=all.equal(cr3,cr1)))
xy1 <- coord_cr(a,c=cr1["c",],r=cr1["r",])
print(xy1)
print(list('in x'=identical(unname(xy1["x",]),b[, "x", drop=TRUE])
          , 'in y'=identical(unname(xy1["y",]),b[, "y", drop=TRUE])))
val1 <- value_xy(a,x=b$jx,y=b$jy)
val2 <- value_xy(a,x=b$jx,y=b$jy)
val3 <- value_cr(a,ind=ind)
val4 <- value_cr(a,c=cr1["c",],r=cr1["r",])
print(val1)
print(list('val1 and val2'=all.equal(val1,val2)
          , 'val2 and val3'=all.equal(val2,val3)
          , 'val3 and val4'=all.equal(val3,val4)
          , 'val4 and val1'=all.equal(val4,val1)))
ps <- pixelsize()
v <- value_ll(ps,lon=180,lat=70)
print(c('True scale'=v/with(ursa_grid(ps),1e-6*resx*resy)))
```

ignorevalue	<i>Extract and assign 'nodata' value of raster images.</i>
-------------	--

Description

Ignored values (*aka* 'nodata') are implemented via NA values, and are optional for raster images in memory. However, to avoid ambiguity for data storage, it is desirable to specify ignored value. "ENVI .hdr Labelled Raster" supports 'nodata' by means of "data ignore value" field in the header file.

Usage

```
ignorevalue(obj)
ursa_nodata(obj)

ignorevalue(obj) <- value
ursa_nodata(obj) <- value
```

Arguments

obj	ursaRaster object.
value	Integer of numeric of length one. Ignored ('nodata') value.

Details

ursa_nodata is synonym to ignorevalue for both *Extract* and *Replace* methods.

The 'nodata' value of raster image obj is specified in the item obj\$con\$nodata.

If values of raster image are in memory then *replace* function ignorevalue<- also changes 'nodata' values to NA values.

Value

Extract function ignorevalue returns value of \$con\$nodata item of ursaRaster object.

Replace function ignorevalue<- returns ursaRaster with modified \$con\$nodata item.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- round(ursa_dummy(nband=1,min=0.500001,max=4.499999))
print(a)
print(as.table(a))
print(ignorevalue(a))
ignorevalue(a) <- NA
```

```
print(as.table(a))
print(ignorevalue(a))
ignorevalue(a) <- 4
print(as.table(a))
print(ignorevalue(a))
print(a)
```

is.na	<i>'No data' values for raster images.</i>
-------	--

Description

The "Extract" function `is.na` creates mask for each band. In this mask value 1L corresponds to NA value in the source image, and value NA corresponds non-missing values in the source image. The "Replacement" function `is.na<-` assigns numerical value for cells with 'no data' value.

Usage

```
## S3 method for class 'ursaRaster'
is.na(x)

## S3 method for class 'ursaRaster'
is.infinite(x)

## S3 method for class 'ursaRaster'
is.nan(x)

## S3 replacement method for class 'ursaRaster'
is.na(x) <- value
```

Arguments

x	Object of class <code>ursaRaster</code>
value	Numeric.

Details

These functions are corresponded to **local** operators of map algebra.

Value

"Extract" functions `is.na`, `is.infinite`, `is.nan` return object of class `ursaRaster`.

"Replacement" function `is.na<-` modifies object of class `ursaRaster`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```

session_grid(NULL)
session_grid(regrid(mul=1/4))
a <- ursa_dummy(nband=2,min=0,max=100)
print(a)
print(is.na(a))
a2 <- ursa_new(nband=2)
print(a2)
print(is.na(a2))
a3 <- a
a3[a3<30 | a3>70] <- NA
print(a3)
print(is.na(a3))
is.na(a3) <- 200
print(a3)

```

legend_align

Align caption position for legend

Description

When multiple panels on the same axis, the different order of values or different units of values may provoke different shifting of values and caption from panels. `legend_align` repairs it by the taking names of classes of the required rasters. The function output is for argument `aling` of [legend_colorbar](#).

Usage

```
legend_align(obj)
```

Arguments

`obj` Object of class `ursaColorTable`, or object of class `ursaRaster`, or list of `ursaColorTable` or `ursaRaster` objects.

Details

The function is defined as:

```
c(unlist(sapply(obj,function(x) names(ursa_colortable(x)))))
```

Value

Character vector.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also[legend_colorbar](#)**Examples**

```

session_grid(NULL)
a <- ursa_dummy(5,mul=1/4,min=-150,max=200)
a[1] <- a[1]*100
a[2] <- -a[2]*10
a[3] <- a[3]/10
a[4] <- a[4]/1000
b <- lapply(a,colorize)
la.top <- legend_align(b[c(1,2)])
la.left <- legend_align(c(b[[1]],b[[3]]))
la.bottom <- legend_align(b[c(3,4)])
la.right <- legend_align(b[c(2,4)])
leg <- vector("list",12)
leg[[1]] <- list("top",2)
leg[[2]] <- list("top",3)
leg[[3]] <- list("bottom",1)
leg[[4]] <- list("bottom",2)
leg[[5]] <- list(2,"left")
leg[[6]] <- list(1,"right")
leg[[7]] <- list(3,"left")
leg[[8]] <- list(2,"right")
leg[[9]] <- list("top",1)
leg[[10]] <- list("bottom",3)
leg[[11]] <- list(1,"left")
leg[[12]] <- list(3,"right")
cl <- compose_design(layout=c(3,3),legend=leg,byrow=TRUE,skip=5)
print(cl)
compose_open(cl)
ct <- compose_panel(b[c(5,1,2,1,4,3,4,5)],decor=FALSE)
L <- 2
Tr <- 2
legend_colorbar(b[1],trim=Tr,las=L,align=la.top,units="top aligned --->")
legend_colorbar(b[2],trim=Tr,las=L,align=la.top,units="<--- top aligned")
legend_colorbar(b[3],trim=Tr,las=L,align=la.bottom,units="bottom aligned --->")
legend_colorbar(b[4],trim=Tr,las=L,align=la.bottom,units="<--- bottom aligned")
legend_colorbar(b[1],trim=Tr,las=L,align=la.left,units="<--- left aligned")
legend_colorbar(b[2],trim=Tr,las=L,align=la.right,units="<--- right aligned")
legend_colorbar(b[3],trim=Tr,las=L,align=la.left,units="left aligned --->")
legend_colorbar(b[4],trim=Tr,las=L,align=la.right,units="right aligned --->")
legend_colorbar(b[5],trim=Tr,las=L,units=" *** not aligned ***")
legend_colorbar(b[5],trim=Tr,las=L,units=" *** not aligned ***")
legend_colorbar(b[5],trim=Tr,las=L,units=" *** not aligned ***")
legend_colorbar(b[5],trim=Tr,las=L,units=" *** not aligned ***")
compose_close()

```

legend_colorbar	<i>Plot colorbar</i>
-----------------	----------------------

Description

Functions draw single color bar outside of maps panels. legend_colorbar (without prefix dot) is a wrapper for non-public .legend_colorbar (with prefix dot)

Usage

```
legend_colorbar(...)  
  
## non-public  
.legend_colorbar(ct, units = "", labels = NA, align = NULL, shift = 1, cex = 1,  
  adj = NA, las = 1, forceLabel = FALSE, lomar = 0, himar = 0,  
  turn = FALSE, useRaster = NA, trim = 0L, abbrev = 24L,  
  opacity = NA, verbose = FALSE)
```

Arguments

... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes. Passed to non-public .legend_colorbar, excepting argument colorbar:

<i>Matched pattern</i> (legend_colorbar)	<i>Argument</i> (.legend_colorbar)	<i>Description</i>
colorbar		Prefix for indirect use (e.g., in display). Separated
(ct)*	ct	<i>See below.</i>
unit(s)*	units	<i>See below.</i>
labels	labels	<i>See below.</i>
align	align	<i>See below.</i>
shift	shift	<i>See below.</i>
cex	cex	<i>See below.</i>
adj	adj	<i>See below.</i>
las	las	<i>See below.</i>
forceLabel	forceLabel	<i>See below.</i>
lomar	lomar	<i>See below.</i>
himar	himar	<i>See below.</i>
turn	turn	<i>See below.</i>
useRaster	useRaster	<i>See below.</i>
trim	trim	<i>See below.</i>
abbrev	abbrev	<i>See below.</i>
opacity	opacity	<i>See below.</i>
verb(ose)*	verbose	<i>See below.</i>

ct ursaRaster object with color table or object of class ursaColorTable. First argument in legend_colorbar; name can be omitted.

units	Argument of class character or expression with matching name "unit(s)*" in legend_colorbar. Text, which is used as a caption for color bars. If character then caption is displayed in bold. Default is "": no caption.
labels	Argument of class integer or character with matching name "labels" in legend_colorbar. If labels is vector of length 1, then it is number of labels at the color bar, else vector of specified values. Default is NA: it means 11 labels for numerical values and 31 labels for categorical values, but this number can be reduced for perpendicular orientation to the axes to prevent label overlapping.
align	Argument of class numeric with matching name "align" in legend_colorbar. The indent for right alignment of labels. May be useful, if two or more color bars are located on the same side, but with different units and order of values. Can be specified by the string of maximal length or via legend_align . Default is NULL: right alignment of each color bar is independent.
shift	Argument of class numeric with matching name "shift" in legend_colorbar. Multiplier for manual correction of labels alignment in the case when automatic alignment is poor. Default is 1: no changes. If shift<1 then labels are shifted to the left. If shift>1 then labels are shifted to the right.
cex	Argument of class numeric with matching name "cex" in legend_colorbar. A numerical value giving the amount by which labels should be magnified relative to the default. Default is 1.
adj	Argument of class numeric with matching name "adj" in legend_colorbar. Adjustment for labels. For labels parallel to the axes, adj=0 means left or bottom alignment, and adj=1 means right or top alignment. Default is NA: for labels parallel to the axes adj=0.5, for labels perpendicular to the axis adj=1 for numeric and adj=0 for character.
las	Argument of values 0, 1, 2, 3 with matching name "adj" in legend_colorbar. The correspondence between directions of axis and labels. The same definition as for par(las=) . Default is 1L.
forceLabel	Argument of class logical with matching name "forceLabel" in legend_colorbar. If TRUE then all labels are plotted regardless their possible overlapping.
lomar	Argument of class numeric, non-negative, with matching name "lomar" in legend_colorbar. Relative shifting of the lower (left or bottom) position of colorbar. Default is 0: the lower position is corresponded to the limit of panel(s). Positive value decreases length of colorbar.
himar	Argument of class numeric, non-negative, with matching name "himar" in legend_colorbar. Relative shifting of the higher (right or bottom) position of colorbar. Default is 0: the higher position is corresponded to the limit of panel(s). Positive value decreases length of colorbar.
turn	Argument of class logical with matching name "turn" in legend_colorbar. Default is FALSE: lower value is on left or bottom, higher value is on right or top. If turn=TRUE, then opposite order.
useRaster	Argument of class logical with matching name "useRaster" in legend_colorbar. Passed as argument useRaster to function image . Default is NA, which is interpreted as TRUE for "cairo" graphical device and as FALSE for "windows" graphical device (see description of argument type in png).

trim	Argument of values 0L, 1L, 2L with matching name "trim" in legend_colorbar. Determines behaviour for plotting marginal labels. If 0L, then marginal labels are displayed as is. If 1L, then marginal labels are shifted inside of color bar to prevent their outcrop to the panel(s) limits. If 2L then outcropped labels are not displayed.
abbrev	Argument of class integer or logical with matching name "abbrev" in legend_colorbar. TRUE is interpreted as default value. FALSE is interpreted as 0L. If positive, then labels are abbreviated, and this argument is passed as argument minlength to function abbreviate : <code>abbreviate(label, minlength=abbrev, strict=TRUE)</code> . If abbreviation is failed (e.g., non-ASCII symbols), the subset is applied.
opacity	Argument of class integer or logical with matching name "opacity" in legend_colorbar. Responses for shading of color bar. If FALSE or 0, then no shading. If TRUE or 1, then shading is forced. Default is NA; if semi-transparence is detected, then shading is applied.
verbose	Argument of class logical with matching name "verb(ose)*" in legend_colorbar. Value TRUE may provide some additional information on console. Default is FALSE.

Details

If units are expression, then possible way for formatting is:

```
units=as.expression(substitute(bold(degree*C)))
```

Value

NULL

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
display(ursa_dummy(1),units="Required 99 labels; displayed less"
      ,colorbar.labels=99,las=3,gridline.trim=FALSE,colorbar.trim=1L)
cname <- c("Apple","Orange","Cherry","Blueberry","Strawberry","Currant")
a <- ursa_dummy(4)
b <- list(colorize(a[1],value=seq(50,200,length=length(cname))
      ,name=cname)#,stretch="category")
      ,colorize(a[2]*10,ramp=FALSE),colorize(a[3]*100),colorize(a[4]/10))
la <- legend_align(b[3:4])
leg <- vector("list",10)
leg[[1]] <- list(1,"left")
leg[[2]] <- list(1,"right")
for (i in seq(4)) {
  leg[[i+2]] <- list("top",i)
  leg[[i+6]] <- list("bottom",i)
}
```

```
compose_open(layout=c(1,4),legend=leg,scale=NA,dev=FALSE) # use 'dev=TRUE' to check layout
compose_panel(b)
legend_colorbar(b[[1]],lomar=20,himar=0) ## "left"
legend_colorbar(b[[4]],labels=c(6,7.5,12,15,20)
,units="Manual set of labels") ## "right"
legend_colorbar(b[[1]],las=2,adj=0.5,turn=TRUE,lomar=6,himar=6
,units="Central adjustment; inverse order") ## ("top",1)
legend_colorbar(b[[2]],cex=0.9
,units="Horizontal labels can be overlapped") ## ("top",2)
legend_colorbar(b[[3]],las=3,align=la
,units="Increased width, but aligned -->") ## ("top",3)
legend_colorbar(b[[4]],las=3,align=la,labels=3
,units="<-- Reduced width, but aligned") ## ("top",4)
legend_colorbar(b[[1]],las=2,adj=0,shift=0.9,turn=FALSE,lomar=2,himar=10
,units="Left adjustment. Non-optimal; shifted") ## ("bottom",1)
legend_colorbar(b[[2]],las=3,adj=0
,units="But right adj. is default for numeric") ## ("bottom",2)
legend_colorbar(b[[3]],labels=99,las=3,trim=2L
,units="Required 99 labels, but displayed less") ## ("bottom",3)
legend_colorbar('Caption from named item'=b[[4]],labels=99) ## ("bottom",4)
compose_close()
```

legend_mtext	Write marginal text
--------------	---------------------

Description

Functions write text outside of maps panels. legend_mtext (without prefix dot) is a wrapper for non-public .legend_mtext (with prefix dot).

Usage

```
legend_mtext(...)

## non-public
.legend_mtext(text = "Annotation", cex = 1)
```

Arguments

... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes. Passed to non-public .legend_mtext, excepting argument mtext:

mtext Prefix for indirect use (e.g., in [compose_legend](#)). Separated by a dot ". ", e.g., mtext.cex=0.85.

Matched pattern (legend_colorbar)	Argument (.legend_colorbar)	Description
mtext		Prefix for indirect use (e.g., in compose_legend). See below.
text	text	

cex	cex	<i>See below.</i>
text	Argument of class character or expression with matching name "text" (or without name) in legend_mtext. Text, which is displayed. If character then text is displayed in bold. Default is "Title/subtitle".	
cex	Argument of class numeric with matching name "cex" in legend_mtext. A numerical value giving the amount by which labels should be magnified relative to the default. Default is 1.	

Details

If text is expression, then possible way for formatting is:

```
text=as.expression(substitute(bold(italic("Omega powered by alpha is ",Omega^alpha))))
```

Value

Returned value of function `mtext` from package **graphics**.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

`compose_legend`
`legend_colorbar`

Examples

```
session_grid(NULL)
a <- ursa_dummy(1,min=-10,max=+30)
compose_open(legend=list("right","top","bottom","left"))
panel_new()
ct <- panel_raster(a)
legend_colorbar(ct)#,units=as.expression(substitute(bold(degree*C))))
legend_mtext("Characters are in bold")
legend_mtext(as.expression(substitute(italic(
  paste("Units can be interpreted as",~degree*C))))),cex=0.7)
legend_mtext(text=as.expression(substitute(italic(paste("Omega powered by alpha is"
  ,~~Omega^alpha))))))
compose_close(execute=!FALSE)
```

local_group

Create single-band raster using statistics of multi-bands raster.

Description

Local operations (mean value, sum of values, median, minimum, maximum) of map algebra for multi-bands `ursaRaster` object.

Usage

```
local_mean(x, cover = 0.5 - 1e-3, weight = NULL, verbose = FALSE, bandname = "mean")
local_sum(x, cover = 0.5 - 1e-3, weight = NULL, verbose = FALSE, bandname = "sum")
local_median(x, cover = 0.5 - 1e-3, verbose = FALSE)
local_min(x, cover = 0.5 - 1e-3, verbose = FALSE)
local_max(x, cover = 0.5 - 1e-3, verbose = FALSE)
local_sd(x, cover = 0.5 - 1e-3, verbose = FALSE)
local_var(x, cover = 0.5 - 1e-3, verbose = FALSE)
local_quantile(x, probs = seq(0, 1, 0.25), type = 7, cover = 0.5 - 1e-3, verbose = FALSE)

## S3 method for class 'ursaRaster'
mean(x, ...)

## S3 method for class 'ursaRaster'
median(x, ...)

## S3 method for class 'ursaRaster'
quantile(x, ...)

# non public
.average(x, cover = 0.5 - 1e-3, weight = NULL, sum = FALSE, verbose = FALSE)
```

Arguments

<code>x</code>	<code>ursaRaster</code> object. In function <code>local_mean</code> and <code>local_sum</code> it is allowed to specify array with 3 dimensions (col, row, band) or (row, col, band)
<code>cover</code>	Numeric. $0 \leq \text{cover} \leq 1$ or > 1 . Quota for NA values in the location for all bands. Quota exceeding leads to recording NA value in the created map. If $\text{code} > 1$ then number of bands. If $0 \leq \text{cover} \leq 1$ then proportion cover to number of bands.
<code>weight</code>	Positive numeric of length equal to number of bands. For <code>local_mean</code> and <code>local_sum</code> only. If specified, then weighted mean or sum are applied. The prior normalization is not required.
<code>sum</code>	Logical. For <code>.average</code> only. If <code>sum=TRUE</code> then fuction returns sum of values else mean value.
<code>probs</code>	Numeric. For <code>local_quantile</code> only. Argument <code>probs</code> , which is passed as argument <code>probs</code> to generic function <code>quantile()</code> .

type	Numeric. For <code>local_quantile</code> only. Argument type, which is passed as argument probs to generic function <code>quantile()</code> .
verbose	Logical. If <code>verbose=TRUE</code> then some output appears in console. Used for debug and benchark.
bandname	Character. Band name for created single-band image.
...	Function mean - arguments, which are passed to <code>local_mean()</code> . Function median - arguments, which are passed to <code>local_median()</code> . Function quantile - arguments, which are passed to <code>local_quantile()</code> .

Details

If for valid output cell value it is required to have at least `m` values not marked as NA, specify quota as `cover=m/nband(x)`.

`local_mean` and `local_sum` are wrapper to non-public function `.average`.

Generic functions `mean`, `median`, `sd` for `ursaRaster` class are implemented via `local_mean`, `local_median`, `local_sd`, respectively.

Value

Double-band `ursaRaster` object for `local_range()`.
Multi-band `ursaRaster` object for `local_quantile()`.
Otherwise, single-band `ursaRaster` object.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

If bands are interpreted as time series, apply `local_stat`

Mean value for image brick `global_mean`

Mean value for each band `band_mean`

Examples

```
session_grid(NULL)
b <- ursa_dummy(nband=7,min=0,max=100,mul=1/16)
b[b<40] <- NA
print(b)
res <- c('mean'=mean(b), 'local_mean'=local_mean(b)
        , 'sum0'=local_sum(b,cover=0), 'sum1'=local_sum(b,cover=1))
print(res)

display(b)
display(res)
```

local_stat	<i>Bundle of statistics, which is applied to each cell of multi-band image.</i>
------------	---

Description

If bands of `ursaRaster` object are interpreted as observations in time, then `local_stat` returns some parameters for time-series analysis. This is a **local** operation of map algebra.

Usage

```
local_stat(obj, time = NULL, cover = 1e-06, smooth = FALSE, verbose = FALSE)
```

Arguments

<code>obj</code>	Object of class <code>ursaRaster</code>
<code>time</code>	Numeric or NULL. If NULL then regression parameters are for regular time-series using position of band in the brick (or, <code>time=seq(obj)</code>). If numeric, then length of time should be equal to number of bands of <code>obj</code> , and time is used to set irregularity for time-series.
<code>cover</code>	Numeric. $0 \leq \text{cover} \leq 1$ or > 1 . Quota for NA values in the location for all bands. Quota exceeding leads to recording NA value in the cell of created band. If <code>code > 1</code> then number of bands. If $0 \leq \text{cover} \leq 1$ then proportion cover to number of bands. Default is $1e-6$.
<code>smooth</code>	Logical. If TRUE then median focal smoothing is applying to created 'slope' band; it is more suitable for visualization. Default is FALSE.
<code>verbose</code>	Logical. Value TRUE provides some additional information on console. Default is FALSE.

Value

Object of class `ursaRaster` with bands:

<code>mean</code>	Mean value in each cell across all bands of source raster.
<code>sd</code>	Standard deviation in each cell across all bands of source raster. Denominator is <code>n</code> .
<code>sum</code>	Sum value in each cell across all bands of source raster.
<code>min</code>	Minimal value in each cell across all bands of source raster.
<code>max</code>	Maximal value in each cell across all bands of source raster.
<code>n</code>	Number of non-NA values in each cell across all bands of source raster (number of observations).
<code>slope</code>	Slope value in each cell across all bands of source raster.
<code>slopeS</code>	Significance of slope value taken with a sign of slope.
<code>RSS</code>	Residual sum of squares.
<code>ESS</code>	Explained sum of squares.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[Local](#) statistics of map algebra, [Group](#) generics for objects of class `ursaRaster`.

Examples

```
session_grid(NULL)
set.seed(353)
session_grid(regrid(mul=1/8))
a <- ursa_dummy(nband=15)
a[a<60] <- NA
cvr <- 12
b <- local_stat(a,cover=cvr)
print(b)
c.mean <- c('<bundle> mean'=b["mean"]
            , 'local_mean'=local_mean(a,cover=cvr)
            , '<generic> mean'=mean(a,cover=cvr))
c.max <- c('<bundle> max'=b["max"]
          , 'local_max'=local_max(a,cover=cvr)
          , '<generic> max'=max(a,cover=cvr))
print(c.mean)
print(c.max)
cmp <- c(mean=b["mean"]-local_mean(a,cover=cvr)
        , sd=b["sd"]-local_sd(a,cover=cvr))
print(round(cmp,12))
d <- as.list(b)
d[["slopeS"]] <- colorize(d[["slopeS"]],stretch="signif")
display(d,blank.density=20,blank.angle=c(-45,45))
```

na.omit

Drop bands which don't have data.

Description

The bands with [band_blank](#) images, are omitted.

Usage

```
## S3 method for class 'ursaRaster'
na.omit(object, ...)
```

Arguments

<code>object</code>	Object of class <code>ursaRaster</code> .
<code>...</code>	Ignored. For consistence with definition of generic function.

Value

Object of class `ursaRaster`, which has no bands without any data.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[band_blank](#)

Examples

```
session_grid(NULL)
session_grid(regrid(mul=1/4))
a <- ursa_new(value=1:3)
print(a)
a[2] <- NA
print(a)
a2 <- na.omit(a)
print(a2)
```

nband

Get number of bands of raster image.

Description

`nband` (`length`) returns number of bands (*layers*, if appropriate in terminology) of `ursaRaster` object.

Usage

```
nband(x)
```

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

Arguments

`x` Object of class `ursaRaster`

Details

`length` for `ursaRaster` object is a synonym for `nband`.

Value

Positive integer of length 1.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[bandname](#) ([names](#) for [ursaRaster](#) object).

Examples

```
session_grid(NULL)
a1 <- pixelsize()
print(a1)
print(nband(a1))
a2 <- c("Band 1"=a1,Band2=a1/2,sqrt=sqrt(a1),NA)
print(a2)
print(nband(a2))
```

open_envi	<i>open_envi file</i>
-----------	-----------------------

Description

open_envi creates object of [ursaRaster](#) class, reads ENVI header file and prepares [connections](#) for ENVI binary file

Usage

```
open_envi(fname, resetGrid = FALSE, headerOnly = FALSE, decompress = !headerOnly,
          cache = 0L, ...)
```

Arguments

fname	Filename; full-name or short-name
resetGrid	Logical. If TRUE then existing base grid (from session_grid) will be overwritten. Otherwise the spatial subsetting will be attempted.
headerOnly	Logical. If TRUE then only reading of ENVI header file without creating connection to binary data; there is no necessary to decompress packed binary in this case. Default is FALSE.
decompress	If ENVI binary file is compressed and you have not to use ENVI values then put decompress=FALSE to avoid useless operation
cache	Integer. Using cache for compressed files. If 0L then cache is not used. If 1L, then cache is used. Any value, which is differed from 0L and 1L, resets cache. Default is 0L.
...	If input file does not exists then these additional arguments will be passed to create_envi function.

Details

open_envi try to find ENVI files (binary and header) and open them. If unsuccessful then function passes ...-arguments to [create_envi](#) function

Value

Returns object of class ursaRaster. Values from ENVI binary are not in memory yet.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[close](#), [create_envi](#)

Examples

```
session_grid(NULL)
a <- pixelsize()
write_envi(a, "example")
a <- open_envi("example")
dir(pattern="^example.*")
ursa_info(a)
close(a)
rm(a)
envi_remove("example")

## additional arguments are enough to create new ENVI file
dir(pattern="^example.*")
a <- open_envi("example", layername=paste0("test", 1:3))
ursa_info(a)
dir(pattern="^example.*")
close(a)
envi_remove("example")
```

open_gdal

Open GDAL file

Description

open_gdal creates object of ursaRaster class, and prepares [connections](#) for data reading.

Usage

```
open_gdal(fname, engine=c("native", "sf", "gdalraster", "vapour"), verbose = FALSE)
```

```
ursa_open(fname, verbose = FALSE)
```


Arguments

fname	Character. Filename; full-name or short-name.
engine	Character. Functionality of which package is used for reading data. This is experimental list, which future depends on evolution of reviewed packages and their availability for partial reading of multiband rasters.
verbose	Logical. verbose=TRUE provides some additional information on console. Default is FALSE.

Details

ursa_open is a synonym to open_gdal. *Generally, both function names are abridged version of ursa_open_dgal.*

open_gdal doesn't read data. Data can be read later using [Extract](#) operator `[]`.

If argument fname is **ENVI .hdr Labelled Raster** then either open_gdal or [open_envi](#) can be used. The former provides external implementation for data reading via GDAL in **rgdal** package.

Value

Returns object of class ursaRaster. Values are not in memory.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[close](#), [open_envi](#), [read_gdal](#).

Examples

```
session_grid(NULL)
# fname1 <- system.file("pictures/cea.tif",package="rgdal")
fname1 <- system.file("tif/geomatrix.tif",package="sf")
message(fname1)
a1 <- open_gdal(fname1)
print(a1)
print(a1[])
close(a1)
# fname2 <- system.file("pictures/test_envi_class.envi",package="rgdal")
fname2 <- tempfile(fileext=".")
a <- ursa_dummy(1,resetGrid=TRUE)
b <- colorize(a[a>91],stretch="equal",name=format(Sys.Date()+seq(0,6),"%A %d"))
write_envi(b,fname2)
message(fname2)
b1 <- open_gdal(fname2)
b2 <- open_envi(fname2)
print(b1)
print(b2)
print(c('The same grid?'=identical(ursa_grid(b1),ursa_grid(b2)))
```

```
        , 'The same data?'=identical(ursa_value(b1[]),ursa_value(b2[])))
close(b1,b2)
envi_remove(fname2)
```

panel_annotation	Add label or annotation to the image panel.
------------------	---

Description

panel_annotation puts an annotation (text label) on the panel with raster image without anchors to any layer. Can be used as captions to image panels.

Usage

```
panel_annotation(...)

# non-public
.panel_annotation(label = expression(), position = "bottomright",
  lon = NA, lat = NA, x = NA, y = NA, cex = 1.0, adjust = 0.5,
  fg = "#000000", bg = "#FFFFFF1F", buffer = 1, fill = "#FFFFFF7F",
  font = par("family"), vertical = FALSE, alpha = 1,
  interpolate = FALSE, resample = FALSE, verbose = FALSE, ...)
```

Arguments

... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes:

<i>Matched pattern</i> (panel_annotation)	<i>Argument</i> (.panel_annotation)	<i>Description</i>
((caption annotation)*)		Logical or integer. Responsible for should annotation
(label text)	label	See below.
pos(ition)*	pos	See below.
lon(gitude)*	lon	See below.
lat(itude)*	lat	See below.
x\$	x	See below.
y\$	y	See below.
cex	cex	See below.
adj(ust)*	adjust	See below.
fg	fg	See below.
bg	bg	See below.
buf(fer)*	buffer	See below.
fill	fill	See below.
font	font	See below.
vert(ical)*	vertical	See below.
(alpha transp(arency)*)	alpha	See below.
interp(olate)*	interpolate	See below.
resample	resample	See below.

verb(ose)*	verbose	See below.
label	Character, expression , or objects of classes <code>array</code> or <code>matrix</code> . Text, symbols or logo for displaying on image panel. Multi-row characters are allowed with delimiter "\n". Default is <code>expression()</code> .	
position	Character keyword or numeric of length 2 in the interval [0,1]. Defines the location of scale bar. If character, then one of the "bottomleft", "bottomright", "topleft", "topright", "left", "right", "bottom", "top", or "center". If numeric then relative position on panel is defined using shift on horizontal and vertical axes from origin in the bottom-left corner. Default is "bottomright".	
lon	Numeric. Longitude for center of annotation's position. Default is NA.	
lat	Numeric. Latitude for center of annotation's position. Default is NA.	
x	Numeric. The horizontal coordinate of the annotation's position in the units of image grid. Default is NA.	
y	Numeric. The vertical coordinate of the annotation's position in the units of image grid. Default is NA.	
cex	Positive numeric. The relative font size for annotation's label. Default is 1. See description of argument <code>cex</code> in text function.	
adjust	One or two values in [0, 1]. Specifies the horizontal (and optionally vertical) adjustment of the labels. See description of argument <code>adj</code> in text function.	
fg	Character. Color name or code for label (texts and symbols). Default is "#000000" (black).	
bg	Character. Color name or code for thin buffer around label's elements. Default is NA, which is interpreted as "transparent" for <i>captions</i> and as "#FFFFFF1F" for <i>annotations</i> .	
buffer	Numeric. The relative width of buffer around label's elements. Default is 1.	
fill	Character. Color name or code for circumscribed rectangle around labels. Default is NA, which is interpreted as "#FFFFFF7F" for <i>captions</i> and as "transparent" for <i>annotations</i> .	
font	Character. Font family. Default is <code>getOption("ursaPngFamily")</code> , which is specified by argument <code>font</code> (or <code>family</code>) in <code>compose_open()</code> .	
vertical	Logical or numeric. Vertical or inclined orientation of label. If FALSE, then horizontal labeling. If numeric then <code>vertical</code> defines text direction in degrees. Limitation: value 1 is interpreted as TRUE. Default is FALSE, which means horizontal text direction.	
interpolate	Logical. Passed as argument <code>interpolate</code> to function rasterImage for logo annotation.	
resample	Logical or numeric. Passed as argument <code>resample</code> to function regrid for logo annotation. Default is FALSE. If TRUE, then resized logo is drawn smoothly.	
alpha	Numeric or character. Level of transparency for logos. If numeric, the either $0 \leq \alpha \leq 1$ or $0 \leq \alpha \leq 255$. If character, then one byte of hexadecimal value "00" $\leq \alpha \leq$ "FF". Default is 1.	
verbose	Logical. Value TRUE may provide some additional information on console. Default is FALSE.	

Details

This function is based on function `text` with adding some decoration elements. For low-level plotting use `layout.text` function, which is equal to function `text` with additional control of image panels.

Since the most of character keywords of position have relation to the boundary of image panel, such annotation is assigned as a *caption* for image panel. Default decoration is shadowed background rectangle, which is implemented by function `rect`.

If location is defined by two-dimensional vector (either relative position inside of image boundaries (pos is numeric of length two), or pair lon, lat, or pair x, y), then such labeling is assigned as an *annotation*. Default decoration is thin buffer around symbols. The implementation is via application of function `text` for small displacements around original position.

The priority of arguments (from higher to lower): 1) pair lon, lat, 2) pair x, y, 3) two-dimensional numeric of pos, 4) character keyword of pos. However, the default annotation is interpreted as a caption.

Value

This function returns NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
## exam no.1 -- direct use
compose_open(layout=c(2,3), legend=NULL, device="cairo")
for (i in seq(6)) {
  panel_new()
  panel_annotation(label=LETTERS, cex=1.5)
  panel_annotation(pos=c(0.7, 0.2)
    , label=paste("panel", paste("no.", i), sep="\n"))
  if (i==1)
    panel_annotation(pos="center")
}
compose_close()

## exam no.2 -- indirect use
display(pixelsize(), scale=2
  , ann.label="FJL", ann.lon=52, ann.lat=80, ann.buffer=1
  , ann.bg="#8F6FFF2F", ann.fill="#FFFF7F9F", ann.font="courier")
```

Description

panel_coastline puts a coastline to the active panel of layout with optional land shadowing. The package provides data for coastline.

Usage

```
compose_coastline(...)

panel_coastline(...)

update_coastline(merge = TRUE)

# not public
.compose_coastline(obj = NULL, panel = 0, col = NA, fill = "transparent",
                    detail = NA, density = NA, angle = NA, land = FALSE,
                    lwd = 0.5, lty = 1, fail180 = NA, verbose = FALSE)

# not public
.panel_coastline(obj, verbose = FALSE)
```

Arguments

... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes:

<i>Matched pattern</i>	<i>Used name</i>	<i>Description</i>
coast(line)*		Logical or integer. Responsible for should coastline be displayed or not. If integer, the indication
(obj)*	obj	See below.
panel	panel	See below.
fill	fill	See below.
detail	detail	See below.
density	density	See below.
angle	angle	See below.
land	land	See below.
lwd	lwd	See below.
lty	lty	See below.
fail180	fail180	See below.
verb(ose)*	verbose	See below.

obj	Objects of the one of the classes Spatial , sfc , ursaCoastLine . The last one is internal structure, which is returned by function .compose_coastline.
panel	Integer vector. Panel for which coastline will be displayed. 0L means that coastline will be displayed for all panels. Default is 0L.
col	Character. Color code/name for coastline. Default is "grey60".
fill	Character. Color code/name for land masking/shadowing. Default is "transparent".

detail	Character keyword. The categorical spatial resolution for coastline. Valid values are "l" (low), "m" (medium), "h" (high), "f" (full). If value is NA, then coastline resolution is selected internally. Default is NA.
density	Numeric. The density of shading lines for land masking/shadowing. If NA then no shading lines are drawn. Default is NA. See density in polygon .
angle	Numeric. The slope of shading lines, given as an angle in degrees (counter-clockwise). If NA then no shading lines are drawn. Default is NA. See angle in polygon .
land	Logical. If TRUE then map's accent is to land, and ocean is masked/shadowed. If FALSE then map's accent is to ocean, and land is masked/shadowed. Default is FALSE.
lwd	Positive numeric. Width of coastline. Default is 1. See lwd in par .
lty	Positive integer. Type (pattern) of coastline. Default is 1L (solid). See lty in par .
fail180	Logical. Patch for correct plot of polygons crossing 180 degree longitude. NA means than decision is taken intuitively. TRUE forces to implement crossing of 180 degree longitude. FALSE forces to not implement crossing of 180 degree longitude. Default is NA.
verbose	Logical. Value TRUE may provide some additional information on console. Default is FALSE.
merge	Logical. Ingored.

Details

compose_coastline forms an object of class `ursaCoastLine`. `panel_coastline` displays object of class `ursaCoastLine`. It is expected higher performance for multi-panel plotting.

If `obj` is `NULL`, then internal data is used. This data is based on simplified polygons of **OpenStreetMap-derived data**. Source data is **licensed** under the Open Data Commons Open Database License (ODbL). The crossing longitude 180 degrees polygons are merged. Removing of small polygons and simplifying of polygons geometry is applied for three levels of details ("l" - low, "i" - interim, "h" - high). For the full ("f") level of details data simplification is not applied.

Coastline data are taken from directory, which is specified by `getOption("ursaRequisite")` with default value `system.file("requisite", package="ursa")`. Package contains data of "l" (low) details level in the file `system.file("requisite/coast-l.rds", package="ursa")`. Data of higher levels can be added using `update_coastline()` function. It is required to specify user's requisite path using `options(ursaRequisite=path/to/user/files)` before loading **ursa**, e.g. in the user's `~/.Rprofile` file. Otherwise, there is a chance that data can not be updated due to 'permission deny' of the system directories. Package **sf** and some of its suggestions are required for data update.

If `detail=NA` then the spatial resolution is selected using CRS boundary and resolution using intuitive approach. If package's database cannot supply required details, then lower resolution is used.

Source coastline data in EPSG:4326 are transformed to CRS projection, extracted using [session_grid](#) function. Coastlines with optional filling of either land or ocean area is interpreted as polygons. If filling is solid (there is no transparency or shading lines (numerical values of arguments density and angle), then coastline plotting is implemented via [polypath](#) function, otherwise [polygon](#) function.

Value

panel_coastline returns NULL

compose_coastline returns of object of class `ursaCoastLine`. It is a list:

<code>coast_xy</code>	Two-column matrix of coordinates in the sessional projection. The polygons are separated by <code>c(NA,NA)</code> rows.
<code>panel</code>	Integer. Panel for coastline displaying. If 0, then coastline is displayed on each panel.
<code>col</code>	See description of argument <code>col</code> .
<code>fill</code>	See description of argument <code>fill</code> .
<code>shadow</code>	If filling is semi-transparent, then it is "alpha" of filling color (argument <code>fill</code>).
<code>land</code>	See description of argument <code>land</code> .
<code>density</code>	See description of argument <code>density</code> .
<code>angle</code>	See description of argument <code>angle</code> .
<code>lwd</code>	See description of argument <code>lwd</code> .
<code>lty</code>	See description of argument <code>lty</code> .

License

Coastal data (land polygons) is distributed under ODbL.

Note

In the versions ≤ 3.7 -19 package **ursa** contained land polygons based on union of "GSHHS_1_L1.shp" and "GSHHS_1_L5.shp" data from Self-consistent Hierarchical High-resolution Geography Database ([GSHHG](#)), Version 2.3.3 (01 November 2014), distributed under the Lesser GNU Public License, Version 3 (29 June 2007).

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- pixelsize()
p1 <- colorize(a[a>500],ramp=FALSE,interval=TRUE)
p2 <- colorize(a,ramp=FALSE,pal=colorRampPalette(c("grey40","grey100"))
              ,verbose=!TRUE,interval=TRUE)
compose_open(layout=c(2,2),legend=list(list(1,"right"),list(2,"left")))
for (i in 1:4) {
  panel_new(col=ifelse(i==2,"white",NA))
  if (i %in% c(3,4))
    panel_raster(p1)
  else if (i %in% 2)
    panel_raster(p2)
  if (i==1)
```

```

    panel_coastline()
    panel_coastline(coast=4,col="#007F00",fill="lightgreen",land=TRUE)
    panel_coastline(coast=3,col="#0000003F",fill="#0000003F")
    panel_coastline(coast=2,col="black",fill="black",density=20
                    ,angle=c(-45,45),lwd=0.25,detail="1")
# panel_graticule(decor=4)
panel_annotation(text=as.character(i))
if (i==1)
  panel_annotation(pos=c(1,1),text="default")
else if (i==2)
  panel_annotation(pos=c(0,1),text="greyscale")
else if (i==3)
  panel_annotation(pos=c(1,1),text="land is shadowed")
else if (i==4)
  panel_annotation(pos=c(0,1),text="ocean is masked")
}
compose_legend(p2,p1)
compose_close()

```

panel_contour

Add colored contour to the image panel

Description

An instrument to overlay multiple rasters on the same image panel. Contour is derived from one band of raster image. The colors (and respective colorbar in legend) is an alternative to contour labeling.

Usage

```
panel_contour(obj, type = "", ...)
```

Arguments

obj	Object of class <code>ursaRaster</code> or <code>NULL</code> . Raster band for contouring. If <code>NULL</code> then contour is not displayed. Default is <code>NULL</code> .
type	Character. Keyword list to describe characteristics of contour, which is interpreted using regular expressions . "label" specifies displaying labels on contour lines. "line" specifies displaying contour lines. "colo(u)*r" specifies displaying colored contour lines. "fill" specifies displaying filled contour lines. Keywords can be combined in a single character, e.g., "fill label" specifies displaying filled contours with labels.
...	Set of arguments, which are recognized via their names (using regular expressions) and classes:

bg Character. Color name or code for contour border. Used for contrast increasing. Semi-transparency and transparency ("transparent") are allowed. Default is "black".

lwd(\\.fg)* Positive numeric. Width of contour line. Default is 1.

lwd\\.bg Positive numeric. Width of the back of contour line. For bordering should exceed foreground width (argument **lwd**). Default is $lwd * 1.5$.

lty Numeric or character. Line type for contour. Default is 1.

(lab)*cex Numeric. Character expansion factor for labels. Default is 0.85.

method Character. Argument, which is passed to [contour](#). Default is "flattest".

expand Numeric. Scale factor (≥ 1) to artificial increasing contour details by means of smoothing of increased image size. Not applicable for images with [color tables](#). Default is NA; smoothing is determined intuitively.

before Logical. Should image reclassification be done before smoothing? Default is FALSE for categorical images and TRUE for numerical images.

cover Numeric. Argument, which is passed to [regrid](#) to control NA values during smoothing. Default is NA; connected to default value of argument **cover** in function [regrid](#).

short Positive integer. Minimal number of points in segments for displaying. Prevents displaying very short segments in the case of high-detailed image. Default is 0: all segments are displayed.

verb(ose)* Logical. Value TRUE may provide some additional information on console. Default is FALSE. Other arguments are used in the function [colorize](#) to produce color tables.

Details

Function [contourLines](#) is used for contouring.

The color table of input raster image is kept. The output panel have one element left, because contours are borders between areas of the same color. It is recommended to use only gradient palettes.

The color table is forced not to be ramp (argument **ramp**=FALSE in the function [colorize](#)) to prevent extra density of contour lines.

The color table is forced to be interval (argument **interval**=1L in the function [colorize](#)) to prevent lost of elements in the palette.

Value

Object of class `ursaColorTable`, which then should be used as an input argument for the colorbar legend (function [legend_colorbar](#)). If there is no argument of class `ursaRaster` then function returns NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also[contourLines](#)[contour](#)**Examples**

```

session_grid(NULL)
a <- pixelsize()
refval <- seq(450,650,by=25)
val <- refval[seq(refval) %% 2 == 1]
ref <- colorize(a,breakvalue=refval,pal.rich=45,pal.rotate=0)
p1 <- colorize(a,breakvalue=val,pal.rich=135,pal.rotate=0)
p2 <- colorize(a,value=val,pal.rich=-15,pal.rotate=0)
p3 <- colorize(a,value=refval)

if (exam1 <- TRUE) {
  compose_open(legend=list(list(1,"left"),list(1,"right")),scale=2)
  panel_new()
  # ct1 <- panel_raster(ref)
  # ct2 <- panel_contour(p2,"colored line",palname="Greens",lwd=15,lwd.bg=0)
  ct2 <- panel_contour(p2,"colored line",pal.rich=240,pal.rotate=0,lwd.fg=15,lwd.bg=0)
  # panel_contour(ref,lwd=0)
  # mysource("contour.R")
  # mycontour(.panel_contour(a),lwd=0)
  if (exists("ct1"))
    compose_legend(ct1,units="raster")
  if (exists("ct2"))
    compose_legend(ct2,units="contour")
  compose_close(bpp=8)
}

if (exam2 <- TRUE) {
  compose_open(layout=c(2,2),byrow=FALSE
    ,legend=list(list(1,"left"),list("bottom",1)
    ,list(1,"right"),list("top",2)
    ,list(2,"right"),list("bottom",2)))

  panel_new()
  panel_raster(ref)
  panel_contour(a)
  panel_new()
  ct0 <- panel_contour(a,"color",value=val,pal.rich=240,pal.rotate=0,lwd=11,lwd.bg=12)
  panel_contour(a)
  panel_annotation(text="no colortable")
  panel_new()
  panel_raster(p1)
  ct1 <- panel_contour(p1,"color",lwd=11,lwd.bg=2)
  panel_contour(a)
  panel_annotation(text="colortable:category")
  panel_new()
  panel_raster(p2)
  ct2 <- panel_contour(p2,"color",lwd=11,lwd.bg=2)
  panel_contour(a)#,cex=0.5)

```

```

    panel_annotation(text="colortable:interval")
    compose_legend(ref,units="reference")
    compose_legend(ct0,units="contour")
    compose_legend(p1,units="raster")
    compose_legend(ct1,units="contour")
    compose_legend(p2,units="raster")
    compose_legend(ct2,units="contour")
    compose_close()
}

if (exam3 <- TRUE) {
  s <- 29
  session_grid(NULL)
  a <- as.ursa(volcano)
  if (FALSE) {
    display(a)
    a2 <- regrid(a,mul=s,cascade=TRUE,verbose=TRUE)
    display(a2)
    session_grid(a)
  }
  compose_open() ## device="windows"
  panel_new()
  ct1 <- panel_raster(a,ramp=FALSE,interval=TRUE)
  ct2 <- panel_contour(a,"label")
  rm(ct2)
  panel_decor()
  if (exists("ct2"))
    legend_colorbar(ct2)
  else if (exists("ct1"))
    legend_colorbar(ct1)
  compose_close()
}

```

panel_decor

Add auxiliary elements to the plotting panel.

Description

panel_decor adds over plot sequentially the followed elements: coastline, gridline, scalebar. Unlike panel_decor, function layout.grid does not add scalebar.

Usage

```
panel_decor(...)
```

Arguments

... Passed to sequence of plotting functions:

- `panel_graticule`. To distinguish the same argument names in different functions it is provided to use prefix “grid.*”, e.g., `grid.col="grey40"`.
- `panel_coastline`. To distinguish the same argument names in different functions it is provided to use prefix “coast.*”, e.g., `coast.col="grey60"`.
- (not applicable for `layout.grid`) `panel_scalebar`. To distinguish the same argument names in different functions it is provided to use prefix “scalebar.*”, e.g., `scalebar.col="black"`.

Details

The sequence of elements is constant. To change order, use direct calling of `panel_graticule`, `panel_coastline`, `panel_scalebar` in any sequence.

Sometimes, for rasters with NA values the followed sequence may be used:

```
panel_coastline(col="transparent",fill="grey80")
panel_raster(a)
panel_coastline(col="grey40")
```

Value

`panel_decor` returns NULL value.
`layout.grid` returns NULL value.

Author(s)

Nikita Platonov <platonov@sebin.ru>

Examples

```
session_grid(NULL)
a <- urda_dummy(nband=1,min=0,max=100)
a[a<30] <- NA
compose_open()
panel_new()
ct <- panel_raster(a)
panel_decor(graticule.col="green4",graticule.lwd=2,scalebar.col="brown")
compose_legend(ct)
compose_close()
```

<code>panel_graticule</code>	<i>Add latitude/longitude or metric grid to the image panel.</i>
------------------------------	--

Description

`panel_graticule` puts a grid on the panel with raster image. If CRS is georeferenced then grid is generated from longitudes and latitudes.

Usage

```

panel_graticule(...)

compose_graticule(...)

# non-public
.compose_graticule(panel = 0L, col = "grey70", border = "grey70", lon = NA, lat = NA,
  lwd = 0.5, lty = 2, marginalia = rep(FALSE, 4), trim = FALSE,
  language = NA_character_, cex = 0.75, verbose = FALSE)

# non-public
.panel_graticule(obj, marginalia = rep(TRUE, 4), verbose = FALSE)

```

Arguments

... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes:

<i>Matched pattern</i>	<i>Used argument</i>	<i>Description</i>
(graticule grid(line)*)		Logical. Responsible for should grid lines be displayed or not.
^(graticule grid(line)*)\$	panel	<i>See below.</i>
(graticule grid(line)\.)*col	col	<i>See below.</i> Default value in <code>compose_graticule</code> depends on
(graticule grid(line)\.)*border	border	<i>See below.</i> Default value in <code>compose_graticule</code> depends on
(graticule grid(line)\.)*lon	lon	<i>See below.</i>
(graticule grid(line)\.)*lat	lat	<i>See below.</i>
(graticule grid(line)\.)*lwd	lwd	<i>See below.</i>
(graticule grid(line)\.)*lty	lty	<i>See below.</i>
(decor margin(alia)*)	marginalia	<i>See below.</i> <code>.compose_graticule</code> and <code>.panel_graticule</code> do
((graticule grid(line)\.)*trim	trim	<i>See below.</i>
(graticule grid(line)\.)*language	language	<i>See below.</i>
(graticule grid(line)\.)*verb(ose)*	verbose	<i>See below.</i>

obj	Objects of the class <code>ursaGridLine</code> . It is internal structure, which is returned by function <code>.compose_graticule</code> .
panel	Integer vector. Panel for which coastline will be displayed. <code>0L</code> means that coastline will be displayed for all panels. Default is <code>0L</code> .
col	Character. Color code/name for grid lines. Default is <code>"grey70"</code> .
border	Character. Color code/name for marginal labels and ticks. Default is <code>"grey70"</code> .
lon	Numeric vector. Set of longitudes for grid. If <code>NA</code> then set of longitudes is formed internally. Default is <code>NA</code> .
lat	Numeric vector. Set of latitudes for grid. If <code>NA</code> then set of latitudes is formed internally. Default is <code>NA</code> .
lwd	Positive numeric. Width of grid line. Default is <code>0.5</code> . See <code>lwd</code> in par .
lty	Positive integer. Type (pattern) of grid line. Default is <code>2</code> (dashed). See <code>lty</code> in par .

marginalia	Logical or integer vectors. Responsible for whether longitudes and latitudes (or metric coordinates) be labelled on the frame of panel with raster image. If logical and TRUE, then labels will be displayed on each open side of panel. If logical and FALSE then labels will not be displayed. If logical of length 4, then labels will be displayed on specific side, where side index is <code>c(bottom, left, top, right)</code> (see description for marginal parameters in par). If argument is a vector of positive integers, then labels for grid lines are plotted only for the specified panels, which sequence is defined in compose_design function and returned from <code>getOption("ursaPngLayout")\$layout</code> . Default is <code>c(TRUE, TRUE, TRUE, TRUE)</code> .
language	Character. Language for longitude and latitude captions. If "ru" then captions are in Russian else in English. Default is NA.
trim	Logical. If grid lines are labelled then <code>trim=TRUE</code> prevents crossing the labels on neighbor perpendicular sides.
cex	Positive numeric. The relative font size for grid lines' labels. Make sense in the case of labels plotting. Default is 0.75.
verbose	Logical. Value TRUE may provide some additional information on console. Default is FALSE.

Details

If not `language="ru"` but environmental variable `LANGUAGE=ru` then labels are in Russian (cyrillics).

Argument `gridline` (or, `grid`) is introduced for unconditional calling of `panel_graticule` inside of high-level functions.

Grid lines can be controlled in high-level plot functions (*e.g.*, [display](#), [compose_plot](#), [display_stack](#), [display_brick](#), [display_rgb](#), *etc.*). To prevent displaying grid lines, use argument `gridline=FALSE` (or `grid=FALSE`). To display grid lines, use argument `gridline=TRUE` (or `grid=TRUE`) and prefix `grid(line)*` (`gridline.*` or `grid.*`) for grid lines' parameters, *e.g.*, `gridline.verb=TRUE`, `grid.col="black"`. If prefix is omitted then arguments with the same names affect in other functions in the part of high-level function.

If grid lines are formed internally, then desirable number of lines for each direction is 3. The design of line density is based on intuition, providing pretty labelling.

If CRS is georeferenced then grid lines are corresponded to longitudes and latitudes. Integer minutes are used to illustrate fractional values of degrees. If precision of minutes is insufficient, then integer values of seconds are introduced. The fractional values of seconds are not used.

Labels are located at the points, where grid lines cross plot margin. Labels are not overlapped along the same side. To prevent overlapping along the same side, labels are shifted or omitted. Argument `trim=TRUE` prevents overlapping labels from neighbor sides via hiding.

Value

Function returns NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```

session_grid(NULL)
## Changing of environmental variables is out of CRAN Policy
## Not run: Sys.setenv(LANGUAGE="ru")

# example no.1
cl <- compose_design(layout=c(2,2),legend=NULL)
session_grid(regrid(lim=3.2*1e6*c(-1,-1,1,1)))
compose_open(cl)
for (i in 1:4) {
  panel_new()
  panel_coastline()
  panel_graticule(decor=TRUE,trim=i %in% c(2:4))
  panel_annotation(text=as.character(i))
  panel_scalebar(scalebar=i==3)
}
compose_close()

# example no.2
session_grid(regrid(lim=1e6*c(-0.5,0.5,1.5,2.5)))
compose_open(layout=c(2,2),legend=NULL,skip=4)
for (i in seq(getOption("ursaPngLayout")$image)) {
  panel_new()
  panel_coastline()
  if (i==1)
    panel_graticule()
  else if (i==2)
    panel_graticule(decor=TRUE,lon=seq(0,360,by=40)[-1],lat=seq(-90,90,by=10))
  else if (i==3)
    panel_graticule(decor=TRUE,lon=seq(0,360,by=20)[-1],lat=seq(-90,90,by=5),
                    ,trim=TRUE)
  else if (i==4)
    panel_graticule(gridline=FALSE)
  panel_scalebar(scalebar=1)
  panel_annotation(text=as.character(i))
}
compose_close()

# example no.3 -- indirect usage
session_grid(NULL)
display(pixelsize(),decor=TRUE,grid.col="green3",coast.col="darkgreen",side=2)

## Changing of environmental variables is out of CRAN Policy
## Not run: Sys.setenv(LANGUAGE="") # reset environmental variable

```

Description

panel_new finishes plotting on previous image panel and starts plotting on next image panel.

Usage

```
panel_new(...)

# non-public
.panel_new(col = "chessboard", alpha = NA, density = NA, angle = NA,
           lwd = 1, lty = 1, asp = NA, mar = rep(0, 4), grid = NULL, verbose = FALSE)
```

Arguments

... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes. Passed to non-public .panel_new. Optional prefix "blank" is used for indirect use. Separated by a dot ".", e.g., blank.fill="transparent".

Pattern (panel_new)	Argument (.panel_new)	Description
(blank\\.)*(^\$ bg fill)	col	See below. Keyword "chessboard" is used by default to produce
(blank\\.)*aphpa	alpha	See below.
(blank\\.)*density	density	See below.
(blank\\.)*angle	angle	See below.
(blank\\.)*lwd	lwd	See below.
(blank\\.)*lty	lty	See below.
(blank\\.)*asp	asp	See below.
(blank\\.)*mar	mar	See below.
(blank\\.)*grid	grid	See below.
(blank\\.)*verb(ose)*	verbose	See below.

col	Character. Color code/name for panel filling/shadowing. Default is "chessboard" (lightened background for controlling transparency) for georeferenced images, and "grey90" for non-projected images.
alpha	Numeric, 0 <= alpha <= 1. Level of transparency. Default is 1, without transparency.
density	Numeric. The density of shading lines for fill/shadowing. If NA then no shading lines are drawn. Default is NA. See density in rect .
angle	Numeric. The slope of shading lines, given as an angle in degrees (counter-clockwise). If NA then no shading lines are drawn. Default is NA. See angle in rect .
lwd	Positive numeric. Width of coastline. Default is 1. See lwd in rect .
lty	Character or positive integer. Type (pattern) of coastline. Default is 1L (solid). See lty in rect .
asp	Positive numeric. The y/x aspect ration. Default is 1. See asp in plot.window .
mar	Positive numeric of length 4. Plot margins. Default is rep(0,4L). See mar in par .

grid	Object of class <code>ursaGrid</code> or converted to it, to define spatial extent and projection for this panel. Default is <code>NULL</code> , which repeats previous state.
verbose	Logical. Value <code>TRUE</code> may provide some additional information on console. Default is <code>FALSE</code> .

Details

Prefix `blank` is introduced for manipulations with `panel_new` inside of high-level functions (e.g., `display`). Prefix `skipping` is the subject for conflict with functions, which use the same name of arguments.

It is required to call `panel_new` for every image panel. First calling starts plotting on the first panel. Second and next callings change image panels.

The panel sequence is set in function `compose_design`, which is called directly or indirectly from `compose_open`, and keeps in the options (access via `getOption("ursaPngLayout")$layout`).

Image background is formed via consecutive call of functions `plot(..., type="n")`, and `rect(...)`.

Value

Function returns `NULL` value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
# example no.1 -- direct use
compose_open(layout=c(1,3),legend=NULL)
panel_new()
panel_annotation(label="Default + Empty")
panel_new(col="#0000FF3F",density=15,angle=45,lwd=3)
panel_decor()
panel_annotation(label="Settings + Grid")
panel_new("#FFFF0040",grid=regrid(expand=0.5))
panel_decor()
panel_annotation(label="Another spatial extent")
compose_close()

# example no.2 -- indirect use
a <- pixelsize()
a <- a[a>560]
display(a,blank.col="#0000FF3F",blank.density=15,blank.angle=45,blank.lwd=3
,coast.fill="#007F005F",coast.density=20,coast.angle=c(-30,60))
```

panel_plot

Add graphical elements to the image panel

Description

Standard functions for plotting from package **graphics** are used for manual adding elements to current plot. These series of functions used that standard instruments with additional controlling the acceptability of plotting.

Usage

```
panel_plot(obj,...)
```

```
panel_box(...)
panel_lines(...)
panel_points(...)
panel_text(...)
panel_abline(...)
panel_polygon(...)
panel_segments(...)
```

Arguments

obj	R object.
...	In panel_plot arguments are passed to function plot. In panel_box arguments are passed to function box. In panel_lines arguments are passed to function lines. In panel_points arguments are passed to function points. In panel_text arguments are passed to function text. In panel_abline arguments are passed to function abline. In panel_polygon arguments are passed to function polygon. In panel_segments arguments are passed to function segments.

Details

If unable to get value TRUE from `getOption("ursaPngPlot")` then plotting is disabled, and any function from this series returns NULL.

Generally, for spatial objects argument `add=TRUE` is used in `panel_plot`.

Value

For spatial objects (simple features from **sf** or spatial abstract classes from **sp**) function `panel_plot` returns object of class `ursaLegend`. It is a list with items, which can be used to as arguments of [legend\(\)](#). This is intermediate step for experimental feature (not ready) to display colorbars on plot panel. For other objects function `panel_plot` returns value of function [plot](#).

Function `panel_box` returns value of function [box](#).
 Function `panel_lines` returns value of function [lines](#).
 Function `panel_points` returns value of function [points](#).
 Function `panel_text` returns value of function [text](#).
 Function `panel_abline` returns value of function [abline](#).
 Function `panel_polygon` returns value of function [polygon](#).
 Function `panel_segments` returns value of function [segments](#).

Note

For plotted elements it is possible to create legend for colors using color bars. No shapes kind and size, no line widths.

To convert object `x` of class `ursaLegend` to object of class `ursaColorTable` please use `ursa_colortable(x)`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[panel_contour](#)

Package **graphics** (`help(package="graphics")`) and functions [plot](#), [box](#), [lines](#), [points](#), [text](#), [abline](#), [polygon](#), [segments](#).

Examples

```
session_grid(NULL)
# require(rgdal) ## 'rgdal' is retired
a <- pixelsize()
g1 <- session_grid()
n <- 12L
k <- 5L
x <- with(g1,runif(n,min=minx,max=maxx))
y <- with(g1,runif(n,min=miny,max=maxy))
panel_plot(x,y) ## plots nothing, because 'compose_open(...,dev=F)' is not called yet
shpname <- tempfile(fileext=".shp")
layername <- gsub("\\.shp$", "", basename(shpname))
if (requireNamespace("sp")) {
  sl <- lapply(seq(k),function(id){
    x <- sort(with(g1,runif(n,min=minx,max=maxx)))
    y <- sort(with(g1,runif(n,min=miny,max=maxy)))
    sp::Lines(sp::Line(cbind(x,y)),ID=id)
  })
  sl <- sp::SpatialLines(sl,proj4string=sp::CRS(ursa_proj(g1)))# ,id=length(sl))
  lab <- t(sapply(sp::coordinates(sl),function(xy) xy[[1]][round(n/2),]))
  lab <- as.data.frame(cbind(lab,z=seq(k)))
  sl <- sp::SpatialLinesDataFrame(sl
    ,data=data.frame(ID=runif(k,min=5,max=9),desc=LETTERS[seq(k)]))
  print(sl@data)
  ct <- colorize(sl@data$ID)# ,name=sldf@data$desc)
```

```

try(writeOGR(sl,dirname(shpname),layername,driver="ESRI Shapefile")) ## 'rgdal' is retired
  spatial_write(sl,shpname)
} else if (requireNamespace("sf")) {
  sl <- lapply(seq(k),function(id) {
    x <- sort(with(g1,runif(n,min=minx,max=maxx)))
    y <- sort(with(g1,runif(n,min=miny,max=maxy)))
    sf::st_linestring(cbind(x,y))
  })
  sl <- sf::st_sfc(sl,crs=as.character(ursa_crs(g1)))
  sl <- sf::st_sf(ID=runif(k,min=5,max=9),desc=LETTERS[seq(k)],geometry=sl)
  print(spatial_data(sl))
  lab <- do.call("rbind",lapply(sf::st_geometry(sl),colMeans))
  lab <- as.data.frame(cbind(lab,z=seq(k)))
  ct <- colorize(sl$ID)
  sf::st_write(sl,shpname)
}
compose_open(layout=c(1,2),legend=list(list("bottom",2)))
panel_new()
panel_decor()
panel_lines(x,y,col="orange")
panel_points(x,y,cex=5,pch=21,col="transparent",bg="#00FF005F")
panel_points(0,0,pch=3)
panel_text(0,0,"North\nPole",pos=4,cex=1.5,family="Courier New",font=3)
panel_new()
panel_decor()
if (exists("sl"))
  panel_plot(sl,lwd=4,col="grey20")
if ((exists("ct"))&&(file.exists(shpname)))
  panel_plot(shpname,lwd=3,col=ct$colortable[ct$index])
if (exists("lab"))
  panel_points(lab$x,lab$y,pch=as.character(lab$z),cex=2)
if (exists("ct"))
  compose_legend(ct$colortable)
compose_close()
file.remove(dir(path=dirname(shpname)
  ,pattern=paste0(layername,"\\.(cpg|dbf|prj|shp|shx)")
  ,full.names=TRUE))

```

panel_raster

Add raster to the image panel

Description

If specified image has 3 or 4 bands, then color composite is plotted on image panel, else the image is plotted regarding to its color table.

Usage

```
panel_raster(...)
```

Arguments

... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes.

1. Passed to [colorize](#).
2. interpreted in this function:

"(^\$|obj)" as obj Object of class `ursaRaster`. Raster band for plotting. Multiple bands are allowed if then can be interpreted as RGB or RGBA.

"useRaster" as useRaster Logical. If TRUE then a bitmap raster is used to plot the image instead of polygons. See argument `useRaster` in function [image](#). Default depends on PNG device (`getOption("ursaPngDevice")`), which is set up in [compose_open](#); it is TRUE for "cairo" device, and FALSE for "windows" device.

"interp(olate)*" as interpolate Logical. Passed as argument `interpolate` to function [rasterImage](#).

"(alpha|transp(aren(cy)*))*" as alpha Numeric or character. Level of transparency. If numeric, the either $0 \leq \alpha \leq 1$ or $0 \leq \alpha \leq 255$. If character, then one byte of hexadecimal value "00" $\leq \alpha \leq$ "FF". If NA, then transparency is used from `colortable`, else transparency of `colortable` is overwritten by alpha. Default is NA.

"verb(ose)*" as verbose Logical. Value TRUE may provide some additional information on console. Default is FALSE.

Details

If `obj` is list of raster images, then `panel_raster` is applied to each item of list, and `colortable` of last item is returned.

If `obj` has 3 or 4 bands then `obj` is interpreted as RGB(A) image.

Function attempts to speed up plotting by reduce image matrix for big rasters.

Value

If argument `obj` has strictly one band, then function returns [color table](#) - object of class `ursaColorTable`, which can be used as an input argument for the colorbar legend (function [legend_colorbar](#)). Otherwise function returns NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
# example no.1 -- direct use
session_grid(regrid(mul=1/32))
dima <- with(session_grid(),c(columns,rows,3))
a <- ursa_new(value=array(runif(prod(dima),min=127,max=255),dim=dima))
p <- colorize(a,pal=c("black","white"),ramp=TRUE,value=0:256)
compose_open(layout=c(2,3),skip=4,legend=list(list("top","full"),list("bottom",2:3)))
```

```

for (i in seq(6)) {
  panel_new()
  if (i<4)
    panel_raster(p[i])
  else
    panel_raster(a,interpolate=i==5)
  panel_decor(col="black",coast=FALSE)
  panel_annotation(c("red","green","blue"
                    ,"interpolate=FALSE","interpolate=TRUE"))
}
legend_colorbar(p,label=seq(0,256,by=16),units="channels")
legend_mtext("color composite")
compose_close()

# example no.2 -- indirect use
ps <- pixelsize(NULL)
display(ps,raster.verb=TRUE)

# example no.3 -- color table for legend
session_grid(NULL)
compose_open()
panel_new()
ct <- panel_raster(ps,pal=terrain.colors)
panel_decor()
compose_legend(ct)
compose_close()

```

panel_scalebar	<i>Add scale bar to the image panel</i>
----------------	---

Description

panel_scalebar puts a scale bar ('box' style) on the panel with raster image.

Usage

```

panel_scalebar(...)

# non-public
.panel_scalebar(position = "bottomleft", w = NA, cex = 0.85,
               col = "#0000002F", bg = "transparent", fill = "#FFFFFF2F",
               language=NA, verbose = FALSE)

```

Arguments

... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes. Passed to non-public .panel_scalebar, excepting argument scalebar:

Pattern (panel_scalebar)	Argument (.panel_scalebar)	Description
(scalebar ruler decor)		Logical or integer. Responsible for should scale bar be displayed.
(scalebar\\.)*pos(ition)*	position	<i>See below.</i>
(scalebar\\.)*w	w	<i>See below.</i>
(scalebar\\.)*cex	cex	<i>See below.</i>
(scalebar\\.)*col	col	<i>See below.</i>
(scalebar\\.)*fill	fill	<i>See below.</i>
(scalebar\\.)*bg	bg	<i>See below.</i>
(scalebar\\.)*language	language	<i>See below.</i>
(scalebar\\.)*verb(ose)*	verbose	<i>See below.</i>

position	Character keyword or numeric of length 2 in the interval [0,1]. Defines the location of scale bar. If character, then one of the "bottomleft", "bottomright", "topleft", "topright", "left", "right", "bottom", "top", or "center". If numeric then relative position on panel is defined using shift on horizontal and vertical axes from origin in the bottom-left corner. Default is "bottomleft".
w	Positive numeric. The length <i>in km</i> of scalebar's right segment. If w=NA then length of segment is defined automatically. Default is NA.
cex	Positive numeric. The relative font size for scalebar's labels. Default is 0.85.
col	Character. Primary fill color for scalebar box and scalebar labels. Default is "#0000002F".
fill	Character. Secondary fill color for scalebar box. Default is "#FFFFFF2F".
bg	Character. Background color for the area of scalebar box and labels. Default is "transparent".
language	Character. Language for longitude and latitude captions. If "ru" then captions are in Russian else in English. Default is NA.
verbose	Logical. Value TRUE may provide some additional information on console. Default is FALSE.

Details

The scalebar has 2 left segments and 2 right segments. Left and right segments are separated by 0. The length of left segments is a half of length of right segments.

Argument scalebar (or, synonym, ruler) is introduced for unconditional calling of panel_scalebar inside of high-level functions.

Default x=0 and y=0 define the "bottomleft" position of scale bar.

If argument scale in the function [compose_open](#) is character, then the length of one segment is exactly 1 cm, and the total length of scalebar is 3 cm.

If not language="ru" but environmental variable LANGUAGE=ru then labels are in Russian (cyrillics).

The length distortions is taken into account for transverse Mercator ("proj=tmerc") projection regarding to location of scalebar.

Scalebar (single occurrence) can be controlled in high-level plot functions (e.g., [display](#), [compose_plot](#), [display_stack](#), [display_brick](#), [display_rgb](#), etc.).

To plot scalebar, use argument `scalebar=TRUE` and prefix (`ruler|scalebar`) (`scalebar.*` or `ruler.*`) for scalebar's parameters, *e.g.*, `scalebar.pos="bottomright"`, `scalebar.cex=0.9`.

Scalebar is not displayed for longlat projection ("`+proj=longlat`"), where units are degrees.

Value

This function returns NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
# example no.1 -- direct usage
a <- colorize(pixelsize())
compose_open(a)
panel_new()
panel_raster(a)
panel_graticule()
panel_coastline()
panel_scalebar()
compose_close()

# example no.2 -- indirect usage
display_rgb(ursa_dummy(nband=3,min=0,max=255),coastline=FALSE
            ,scalebar=TRUE,scalebar.col="white",scalebar.fill="black")

# example no.3 -- for paper copy
a <- colorize(pixelsize(),breakvalue=seq(400,650,by=50),pal=c("gray90","gray30"))
compose_open(scale="1:95000000",dpi=150,device="cairo",family="serif")
compose_plot(a,units=expression(km^2)
            ,graticule=TRUE,coastline=FALSE,scalebar=TRUE,scalebar.pos=c(1,1))
compose_close(bpp=8)

# example no.4 -- length distortion in the Transverse Mercator projection

a1 <- regrid(setbound=c(10,65,71,83),dim=c(100,100),crs=4326)
a2 <- polygonize(ursa_bbox(a1))
a3 <- spatial_transform(a2,3857)
a4 <- regrid(setbound=spatial_bbox(a3),res=20000,crs=spatial_crs(a3))
compose_open(legend=NULL)
panel_new("white")
panel_coastline(fill="#00000010",detail="1")
# panel_graticule()
for (p in c("bottom","center","top"))
  panel_scalebar(pos=p,w=500)
compose_close()
```

panel_shading	<i>Shaded overlay by image mask</i>
---------------	-------------------------------------

Description

This specific function is designed to illustrate linear slope and areas of statistically significant slope on the same panel, however can be used commonly for shading by raster mask.

Usage

```
panel_shading(obj, level = NA, col = NULL, density = 25, angle = c(-45, 45),
              lwd = 1, lty = 1, verbose = TRUE)
```

Arguments

obj	Object of class <code>ursaRaster</code> .
level	Positive numeric. Threshold for obj reclassification { <code>obj<(-level) obj>(+level)</code> }. If NULL then mask is created from non-NA values of obj. Default is NULL.
col	<code>ursaColorTable</code> (<code>ursaRaster</code> with color table) or character. Color for shading lines (grid). If object of class <code>ursaColorTable</code> . Two colors on the limits of color vector are extracted to separate source values <code><=(-level)</code> and <code>>=(+level)</code> .
density	Numeric. The density of shading lines, in lines per inch. Default is 25. See description of argument density in function polygon .
angle	Numeric. The slope of shading lines, given as an angle in degrees (counter-clockwise). Default is vector of length two <code>c(-45, 45)</code> . See description of argument angle in polygon function.
lwd	Numeric. Line width for shading. Default is 1. See description of lwd in par function
lty	Numeric or character. Line type for shading. Default is 1. See description of lty in par function.
verbose	Logical. If TRUE then progresss bar is appeared. Default is TRUE.

Details

Values of input obj is reclassified to raster mask: { `values<=(-level)` OR `values>=(+level)` }. For common use, select appropriate level and, if necessary, reclassify obj prior.

Color limits are extracted using [range](#) function.

Raster images can be used for colored shading using alpha argument of [panel_raster](#) function, e.g. `panel_raster(a, alpha=3/4)`

Value

NULL

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```

session_grid(NULL)
if (first.example <- TRUE) {
  session_grid(NULL)
  session_grid(regrid(mul=1/8))
  ps <- pixelsize()
  compose_open()
  ct <- compose_panel()
  panel_shading(ps>1.1*global_mean(ps),angle=90)
  compose_legend(ct)
  compose_close()
}
if (second.example <- TRUE) {
  session_grid(NULL)
  a <- ursa_dummy(nband=15,mul=1/8)
  b <- local_stat(a)
  compose_open()
  lev <- 0.90
  d <- as.matrix(b["slopeS"],coords=TRUE)
  e <- contourLines(d,levels=c(-lev,lev))
  p <- list(significance.raw=colorize(b["slopeS"]))
    ,significance.formatted=colorize(b["slopeS"],stretch="significance")
    ,slope=colorize(b["slopeS"]))
  p <- c(p,rep(p[3],3))
  names(p)[c(3,4,5)] <- c("Slope and shaded significance"
    ,"Slope and contoured significance"
    ,"Slope and 'contourLines'")
  compose_open(p,layout=c(2,NA),byrow=FALSE)
  compose_panel(p[1])
  compose_panel(p[2])
  compose_panel(p[3])
  panel_shading(b["slopeS"],level=lev)
  compose_panel(p[4])
  panel_contour(b["slopeS"],value=c(-lev,lev))
  compose_panel(p[5])
  lapply(e,panel_polygon)
  compose_panel(p[6])
  ct <- panel_contour(b["slopeS"],"color"
    ,value=c(-0.99,-0.95,-0.9,-0.5,0.5,0.9,0.95,0.99))
  compose_legend(c(head(p,-1),'(Colorbar for contours) '=list(ct)),las=3)
  compose_close()
}

```

pixelsize

The actual size of each grid cell with considerable distortion in area of map projection.

Description

This function helps to calculate size of pixels in the unit of area (squared km) for zonal statistics with taking into account distortion in area for classes of projections.

Usage

```
pixelsize(obj, verbose = FALSE)
```

Arguments

obj	Either ursaRaster object or ursaGrid object or NULL or missing .
verbose	Logical. Value TRUE may provide some additional information on console. Default is FALSE.

Details

pixelsize() is applied to coordinate reference system (grid) of ursaRaster object or to [raster grid](#) directly. If argument obj is [missed](#), then [session grid](#) is used.

Currently, only *Stereographic* ("+stere" in PROJ.4 notation), *Mercator* ("+merc"), and *Lambert Azimuthal Equal Area* ("+laea") classes of map projections are implemented, though the last one (LAEA) has no distortion in area.

Value

Object of class ursaRaster, single-band. If size of cell is more than 10e5 square meters, then the unit is squared kilometers (band name is "Pixel Size (sq.km)") else squared meters (band name is "Pixel Size (sq.m)").

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
pixelsize()

## internet connection is required; access was tested on 2018-06-04
invisible({
  dpath <- file.path("ftp://sidacs.colorado.edu/pub/DATASETS"
                    , "nsidc0081_nrt_nasateam_seaice/north")
  dst <- tempfile(fileext=".bin")
  isOK <- FALSE
  d3 <- Sys.Date()
  for (i in seq(5)) {
    src <- file.path(dpath, format(d3, "nt_%Y%m%d_f18_nrt_n.bin"))
    a <- try(download.file(src, dst, mode="wb"))
    if ((is.integer(a)) && (a==0)) {
      isOK <- TRUE
      break
    }
  }
})
```

```

    }
    d3 <- d3-1
  }
  if (isOK) {
    g1 <- regrid(bbox=c(-385,-535,375,585)*1e4,res=25*1e3
                 ,crs=paste("+proj=stere +lat_0=90 +lat_ts=70 +lon_0=-45"
                             ,"+k=1 +x_0=0 +y_0=0 +a=6378273 +b=6356889.449"
                             ,"+units=m +no_defs"))
    session_grid(g1)
    b <- readBin(dst,integer(),size=1L,n=136492L,signed=FALSE)
    ice <- ursa_new(value=tail(b,-300))
    ice[ice>251] <- NA ## keep Pole
    ice[ice==251] <- 250 ## consider 100% ice at Pole
    ice <- ice/2.5 ## uncategorize
    ice[ice<15] <- 0 ## not ice, if less 15%
    ice[ice>0] <- 100
    extent1 <- band_sum(ice*1e-2*ursa(ice,"cell")^2*1e-6)*1e-6
    extent2 <- band_sum(ice*1e-2*pixelsize(ice))*1e-6
    message(paste("Near real-time Arctic sea ice extent (NASA Team algorithm, NSIDC)"))
    message(sprintf("   Direct area calculation:           %5.2f*1e6 km^2.",extent1))
    message(sprintf("   Distortion in area is corrected: %5.2f*1e6 km^2.",extent2))
  }
  else
    message("It is failed to get sea ice concentration data.")
})

```

plot

Simple display of raster images

Description

Function `image` for `ursaRaster` object calls generic function `image`.
 Function `plot` for `ursaRaster` object calls function `filled.contour`.
[Color tables](#) are supported.

Usage

```
## S3 method for class 'ursaRaster'
plot(x, ...)
```

```
## S3 method for class 'ursaRaster'
image(x, ...)
```

Arguments

`x` Object of class `ursaRaster`
`...` Other parameters. Are passed to or `filled.contour` or to generic function `image`.

Details

Usage of both these functions is justified for low-level control of plotting. It is recommended to use high-level function [display](#). It is flexible and power instrument for raster images visualization.

Function [as.list](#) for `ursaRaster` object transforms single band of raster image to a suitable object for plotting via function [image](#) from package **graphics**

Value

Returned value from [image](#) or [filled.contour](#) (both functions are in the package **graphics**)

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[display](#)

Examples

```
session_grid(NULL)
a <- pixelsize()
plot(a,asp=1)
image(a,asp=1)
b <- colorize(a,ncolor=15)
plot(b,asp=1)
image(b,asp=1)
```

`polygonize`

Raster to vector (polygon) conversion.

Description

Representing each raster cell as a polygon. In comparison to common GIS raster to vector conversion, where neighbor cells with the same value are combined to the single polygon, the number of output polygons is equal to number of non-NA values.

Usage

```
polygonize(obj, fname, engine = c("native", "sf"), verbose = NA, ...)
```

Arguments

<code>obj</code>	Object of class <code>ursaRaster</code> .
<code>fname</code>	Missing or character. If specified, then ESRI Shapefile is created. Default is missing .
<code>engine</code>	Character keyword from list <code>c("native", "sf")</code> . Define package with tools for creating spatial data. If suggested packaged " sp " is loaded or can be loaded from default location, then "sp" is added to this list. If <code>engine="sf"</code> , then functions from package sf are used. If <code>engine="sp"</code> , then functions from package sp are used. If <code>engine="native"</code> , then appropriate package is used based on loaded namespaces before.
<code>verbose</code>	Logical. If TRUE then conversion is attended by progress bar. Default is NA; it means TRUE for <code>engine="sp"</code> and FALSE for <code>engine="sf"</code> .
<code>...</code>	Additional arguments, which are passed to internal function for writing ESRI Shapefile.
<code>compress</code>	Logical. Should output ESRI Shapefile files be compressed by zip? Default is FALSE.

Details

Some GIS software (e.g., QGIS) has broad tools for display vector data. Excepting choroplets, it is assumed that visualization of each cell separately is more attractive than displaying of polygons with different forms, which are produced, for example, by GDAL conversion utility [gdal_polygonize.py](#).

Value

If missing `fname` and tools from **sp** then object of class "SpatialPolygonsDataFrame" (package **sp**).
 If missing `fname` and tools from **sf** then object of class "sf" with geometry of class "sfc_POLYGON" (package **sf**).
 If `fname` is specified, then NULL.

Note

Implementation is very slow even for moderate image size. Use progress bar (`verbose=TRUE`) to control this process.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- ursa_dummy(mul=1/16)
a <- a[a>100]
print(a)
print(band_mean(a))
b2 <- polygonize(a,engine=ifelse(requireNamespace("sp"),"sp","sf"))
print(class(b2))
```

```
print(colMeans(spatial_data(b2),na.rm=TRUE))
str(e1 <- spatial_bbox(a))
str(e2 <- spatial_bbox(b2))
print(as.numeric(e1))
print(as.numeric(e2))
```

read_envi

Read ENVI .hdr Labelled Raster file to memory

Description

Reads all or several bands of ENVI .hdr Labelled Raster file from disk to memory.

Usage

```
read_envi(fname, ...)
```

Arguments

fname	Character. Filename for ENVI .hdr Labelled Raster file.
...	For read_envi: Set of arguments, which are recognized via their names (using regular expressions) and classes. In the case of grids mismatch some arguments (e.g., resample) are passed to regrid function.
(subset)*	<i>Name can be omitted.</i> Integer or character. If integer, then indices of bands, either positive or negative. Positive indices are for included bands, negative indices for omitted bands. If character, then either sequence of band names or regex string. By default (subset=NULL), function reads all bands.
(ref)*	<i>Name can be omitted.</i> ursaRaster or ursaGrid object. Reference grid for raster image resizing. By default (ref=NULL) there is no resizing.
(nodata ignorevalue)	Numeric. Value, which is ignored. By default (nodata=NaN) ignore value is taken from ENVI metadata (*.hdr or *.aux.xml).
reset(Grid)*	Logical. If TRUE, then session grid is ignored, and new session grid is assigned from input file. If FALSE, then input file is nested in the session grid.
cache	Integer. Using cache for compressed files. 0L - do not use cache, 1L - use cache; any other value resets cache. Default is FALSE.
verb(ose)*	Logical. Value TRUE may provide some additional information on console. Default is FALSE.

Details

Function read_envi is designed to one-time reading (from disk to memory) ENVI .hdr Labelled Raster file. For multiple access to disk (by chunks), use followed construction:

```

a <- open_envi(fname)
d1 <- a[condition_1]
d2 <- a[condition_2]
...
close(a)

```

In this case, the connection keeps open. The gain is more effective for compressed binary files.

Value

Object of class `ursaRaster`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[open_envi](#), [Extract](#) method [\[](#) for `ursaRaster` object, [close_envi](#).

[read_gdal](#) uses GDAL ([rgdal](#)) to read ENVI .hdr Labelled Raster file.

Examples

```

session_grid(NULL)
fname <- tempfile()
a <- ursa_dummy()
bandname(a) <- c("first", "second", "third")
write_envi(a, fname, compress=FALSE)

print(read_envi(fname))
print(read_envi(fname, c(1, 3)))
print(read_envi(fname, -c(1, 3)))
print(read_envi(fname, c("first", "third")))
print(read_envi(fname, "iR"))

print(session_grid())
g <- regrid(session_grid(), mul=1/2.3)
b <- read_envi(fname, ref=g)
print(session_grid())
print(b)

envi_remove(fname)

```

read_gdal	<i>Read GDAL supported raster files.</i>
-----------	--

Description

read_gdal creates `ursaRaster` object from GDAL supported raster files using functions from packages with low-level raster reading.

Usage

```
read_gdal(fname, resetGrid = TRUE, band = NULL,  
          engine = c("native", "sf"),  
          verbose = FALSE, ...)
```

```
ursa_read(fname, verbose = FALSE)
```

Arguments

fname	Character. GDAL supported raster file name.
resetGrid	Logical. If TRUE then new sessional grid is based on opened raster image. Default is TRUE
band	Character (regular expression) or integer.
engine	Character. Functionality of which package is used for reading data. This is experimental list, which future depends on evolution of reviewed packages and their availability for partial reading of multiband rasters.
verbose	Logical. Value TRUE may provide some additional information on console. Default is FALSE.
...	Ignored.

Details

ursa_read is simplified implementation of gdal_read.

The composite GDAL formats (e.g., [NetCDF: Network Common Data Format](#), [HDF5: Hierarchical Data Format Release 5](#)) are likely unsupported.

read_gdal uses functions from other other packages. It's a wrapper.

Category names and color tables are supported.

Value

Object of class [ursaRaster](#).

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[as.ursa](#) is an alternative call for GDAL raster files import.

Examples

```
session_grid(NULL)

# rgdal::gdalDrivers()
if (requireNamespace("sf"))
  print(sf::st_drivers())
if (file.exists(Fin1 <- system.file("gdal/gdalicon.png", package="sf"))) {
  a1 <- read_gdal(Fin1)
  print(a1)
  display(a1)
}
Fin2 <- tempfile(fileext=".")
a <- ursa_dummy(1, resetGrid=TRUE)
b <- colorize(a[a>91], stretch="equal", name=format(Sys.Date()+seq(0,6), "%A %d"))
write_envi(b, Fin2)
b1 <- read_gdal(Fin2)
b2 <- read_envi(Fin2, resetGrid=TRUE)
envi_remove(Fin2)
print(c('same colortable?'=identical(ursa_colortable(b1), ursa_colortable(b2))))
print(ursa_colortable(b1))
print(as.table(b1))
print(c('same values?'=identical(ursa_value(b1), ursa_value(b2))))
print(c('same grid?'=identical(ursa_grid(b1), ursa_grid(b2))))
if (requireNamespace("sf")) {
  p1 <- sf::st_crs(ursa_crs(b1))
  p2 <- sf::st_crs(ursa_crs(b2))
  print(c('same proj4string for CRS?'=identical(p1$proj4string, p2$proj4string)))
  print(c('same WKT for CRS?'=identical(p1$Wkt, p2$Wkt)))
  ursa_crs(b1) <- ursa_crs(b2)
  print(c('after same CRS, same grid?'=identical(ursa_grid(b1), ursa_grid(b2))))
}
display(b1, detail="1")
```

reclass

Reclassify specific values of image

Description

This is look-up table reclassification: the destination value is found for each source value.

Usage

```
reclass(obj, dst = NULL, src = NULL, sparse = FALSE, ...)
```

Arguments

<code>obj</code>	Object of class <code>ursaRaster</code> or <code>ursaColorTable</code> .
<code>dst</code>	Object of class <code>ursaRaster</code> , or object of class <code>ursaColorTable</code> , or numeric vector. If numeric, then the desired destination set of values, else reference object for reclassification; this object should have numerical values of categories.
<code>src</code>	Numerical vector, but allowed using with numerical vector of <code>dst</code> and <code>length(src)==length(dst)</code> . Source set of values.
<code>sparse</code>	Logical. If image has a lot of NA values then <code>sparse=TRUE</code> may speed up transformation. Default is <code>FALSE</code> .
<code>...</code>	Other arguments are used for classification in the function <code>colorize</code> .

Details

If `dst` is numeric vector, then the source value have to be specific, without any ranges. It is required the equality lengths of `src` and `dst`. If image has `color table` then function tries reconstruct `dst` from names of `categories`.

This function can be used for data compression for storage, e.g. for distribution or interchange.

Value

If `obj` is object of class `ursaColorTable` then numeric vector of categories' centers.

If `dst` is numeric, then object of class `ursaRaster` without `color table`.

If `dst` is `ursaColorTable` then object of class `ursaRaster` (NA values) in `color table`.

If `dst` is `NULL` then object of class `ursaRaster` with empty color names (NA values) in `color table`.

Note

There were no a lot of tests how GIS software reads "ENVI .hdr Labelled Raster" files with color tables without color values (only *categories*). At least, **GDAL** recognizes categories (`gdalinfo` utility).

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

The reclassification from interval source values to specific destination values is used in `colorize`.

Examples

```
session_grid(NULL)
# example no.1 manual classification
a <- as.ursa(round(matrix(runif(100,min=0.5,max=3.5),ncol=10)))
print(as.table(a))
b <- reclass(a,src=c(3,1,2),dst=round(runif(3),2))
print(as.table(b))
```

```

print(c(src=a,dst=b))

# example no.2 -- similarity to other object
session_grid(NULL)
a <- ursa_dummy(nband=2,min=-1,max=1)
print(a)
b1 <- colorize(a[1],value=seq(-1,1,length=21),pal.rich=240,pal.rotate=0)
b2 <- reclass(a[2],b1)
b3 <- reclass(a[2],ursa_colortable(b2))
b <- c(b1,b2,b3)
print(reclass(b))

# example no.3 -- compression with data lost
a <- pixelsize(NULL)
b <- reclass(a,byte=TRUE,tail=0) ## try 'byte=FALSE'
a2 <- reclass(b)
res <- c(source=a,as_category=a2,difference=a-a2)
print(res)
message(paste("RMS error: ",format(sqrt(band_sum(res[3]^2)/band_n(res[3])))))
prefix <- names(res)[1:2]
fname <- file.path(tempdir(),paste0(prefix,".envi"))
s <- data.frame(object.size=sapply(list(a,b),object.size))
rownames(s) <- prefix
print(s)
write_envi(a,fname[1])
write_envi(b,fname[2])
f <- file.info(dir(path=tempdir()
                    ,pattern=paste0("(",prefix,")\\.(envi|hdr)",sep="|")
                    ,full.names=TRUE))[,"size",drop=FALSE]
colnames(f) <- "file.size"
print(f)
envi_remove(fname)

```

regrid

Change raster image resolution and extent

Description

General function to change parameters of cells under the same geographical projection. It is implemented via raster resampling to the new grid.

Usage

```

regrid(x, ...)

## non-public
.regrid(grid = NULL, mul = NA, res = NA, resx = NA, resy = NA, setbound = NA,
        columns = NA, rows = NA, dim = NA, bbox = NA, expand = NA,
        minx = NA, miny = NA, maxx = NA, maxy = NA, cut = NA, proj4 = NA, crs = NA,
        border = 0, zero = c("keep", "node", "center"), raster = FALSE, tolerance = NA,

```

zoom = NA, rotation = NA, adjust = c(0.5, 0.5), verbose = FALSE, ...)

Arguments

x	Object of class <code>ursaRaster</code> .
...	- Arguments, which are passed to non-public <code>.regrid</code> to define parameters of new grid. - Set of arguments, which are recognized via their names (using regular expressions) and classes: <code>^reset(Grid)*</code> Logical. Whether new grid will be defined as a sessional parameter? If TRUE then returned raster defines new sessional grid. If FALSE then session grid is not changed. Default is TRUE. <code>resample</code> Logical or positive numeric. The range of aggregation in the units of cell area. If 0 or FALSE then "nearest neighbor" value is used. The <code>resample>0</code> defines the side of rectangular area in proportion to cell size; and aggregation of adjacent cells is weighted in proportion to overlapping parts of cells. Default is 1 (or, equally, TRUE); it means that value of output cell is weighted mean of values of overlapped input cells in proportion of overlapping of output cell by input cells. <code>cover</code> Positive numeric in the range [0, 1]. The maximal fraction of NA values in adjusted input cells for the rule to write NA value to the output cell. Default is 0.499. <code>cascade</code> Logical. Option to get more smooth results. If TRUE and <code>resample>2</code> then <code>resize</code> function is applied sequentially with argument <code>resample<=2</code>. <code>verb(ose)*</code> Logical. Value TRUE may provide some additional information on console. Default is FALSE.
grid	Reference ursaGrid or <code>ursaRaster</code> object. If missing then reference grid is obtained from sessional grid <code>session_grid()</code>
mul	numeric of length 1. Multiplication for changing image size by means of changing of cell size (1/mul). <code>mul>1</code> decreases cell size, <code>mul<1</code> increases cell size
res	numeric of length 1 or 2. New grid size by horizontal and vertical axes. If length is 1 then the same grid size for both axes.
resx	Positive numeric of length 1. New grid size by horizontal axis.
resy	Positive numeric of length 1. New grid size by vertical axis.
setbound	numeric of length 4. Change units of spatial extension and define new spatial extension (boundary box) in the notation <code>c(minx,miny,maxx,maxy)</code> .
columns	Positive integer of length 1. Number of columns/samples in the case of definition of new spatial extension (setbound is non-NA).
rows	Positive integer of length 1. Number of rows/lines in the case of definition of new spatial extension (setbound is non-NA).
dim	Positive integer of length 2. Dimension of raster image in the notation <code>c(rows, columns)</code> (or, <code>c(lines, samples)</code>) in the case of definition of new spatial extension (setbound is non-NA).

bbox	numeric of length 4. New spatial extension (boundary box) in the notation c(minx,miny,maxx,maxy) in the same units of existing spatial extension.
minx	numeric of length 1. New value for left boundary.
miny	numeric of length 1. New value for bottom boundary.
maxx	numeric of length 1. New value for right boundary.
maxy	numeric of length 1. New value for top boundary.
cut	numeric of length 4. Vector (left, bottom, right, top) in CRS units for extent expand.
border	numeric of length 1 or 4. If length 4, then vector (bottom, left, top, right) in cells for extent expand. If length <4, then value is repeated for length 4.
proj4	character of length 1. New projection string in the PROJ.4 notation
crs	character of length 1. The synonym to proj4.
expand	numeric of length 1. Multiplier of boundary box.
raster	logical. Should return blank ursaRaster object instead of ursaGrid object? See 'Value' section
zero	character. Define central cell position relative to zero coordinates. If value is "keep", then central cell position is without changes. If value is "node", then zero coordinates are on the crossing of cell borders. If value is "center", then zero coordinates are in the center of central cell. <i>Currently is not implemented. If grid is consistent, then value "keep" is used, else "node".</i>
tolerance	numeric. Threshold for comparison float point numerics. Required for internal check of grid consistence. Default is NA; value .Machine\$double.eps multiplied on maximal value of coordinates is used.
zoom	numeric. Tweak for simultaneous change of expand and mul: expand=zoom, mul=1/zoom. Default is NA.
rotation	numeric. Degrees to specify grid rotation. Rotation is around grid central point. Values NA and 0 mean no rotation. Default is NA.
adjust	numeric of length 2. Dragging horizontal and vertical expansion for argument expand. Value 0.0 means expansion to the left or to the bottom. Value 1.0 means expansion to the right or to the top. Value 0.5 means equal horizontal or vertical expansion. Default is c(0.5, 0.5).
verbose	Reporting via message about violation and restoration of coordinate grid regularity after non-consistence usage of parameters.

Details

Generally, argument `resample` sets a rectangular region. The area of this region is in proportion to area of output cell. Argument `resample` is the value of this proportion. Each cell is interpreted as a set of adjoining rectangular figures. The value of output cells is a weighted mean of that input cells, which fall into rectangular region. The weights are defined as an partial area inside of rectangular region.

Function implements "nearest neighbor" resampling method if argument `resample=0` (or, `resample=FALSE`). If `resample=1` (or, `resample=TRUE`) and both input and output rasters have the same cell size, then resampling method is "bilinear interpolation".

Expand raster *x* to 3 times with cell repeating: `regrid(x,mul=3,resample=FALSE)` ## nearest neighbor;
 Expand raster *x* to 3 times with cell aggregation: `regrid(x,mul=3,resample=TRUE)` ## bilinear interpolation;
 Contract raster *x* to 3 times without cell aggregation: `regrid(x,mul=1/3,resample=FALSE)` ## nearest neighbor;
 Contract raster *x* to 3 times with cell aggregation: `regrid(x,mul=1/3,resample=TRUE)` ## weighted mean;
 Low-pass filtering by 3 x 3 window size: `regrid(x,resample=3*3)` ## see [focal_mean](#)

However, simple contraction `regrid(x,mul=1/2,resample=FALSE)` is implemented as contraction with aggregation (`regrid(x,mul=1/2,resample=FALSE)`), because centers of output cells are located in the nodes (crossing of boundaries of input cells).

It seems that for categorical rasters parameter `resample=0` is more suitable, because nearest neighboring does not introduce new values to output raster, excepting coincidence of input cells' nodes and output cell centers.

Usage of `proj4` argument specifies only desirable PROJ.4 string and does not do reprojection.

The violation of grid regularity is due to columns and rows of image should be integer. The restoration of grid regularity is realized by spatial extension (boundary box) expansion.

Value

`regrid` returns object of class `ursaRaster`.

Return value of non-public function `.regrid` depends on logical value of `raster` argument. If `raster=FALSE` then `.regrid` returns new grid without any change of sessional grid. If `raster=TRUE` then `.regrid` returns blank image and changes sessional grid.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[regrid](#), [focal_mean](#)

Examples

```
session_grid(NULL)
print(g1 <- session_grid())
print(g2 <- regrid(g1,mul=2))
print(g3 <- regrid(g1,res=50000,lim=c(-1200000,-1400000,1600000,1800000)))
print(g4 <- regrid(g1,res=50000,lim=c(-1200100,-1400900,1600900,1800100),verbose=TRUE))
print(g5 <- regrid(g1,mul=1/4))
print(g6 <- regrid(g1,mul=1/4,cut=c(-1,-2,3,4)*25000))
print(g7 <- regrid(g1,mul=1/4,expand=1.05))
print(session_grid()) ## equal to 'g1'
print(a <- regrid(g1,mul=1/4,border=3,raster=TRUE))
print(session_grid()) ## not equal to 'g1'

session_grid(NULL)
```

```

'.makeRaster' <- function(nc=6,nr=8) {
  as.ursa(t(matrix(runif(nc*nr,min=0,max=255),ncol=nc,nrow=nr)))
}
session_grid(NULL)
a <- .makeRaster(12,18)
expand <- 1/3
a1 <- regrid(regrid(a,mul=expand,resample=FALSE),a,resample=FALSE)
a2 <- regrid(regrid(a,mul=expand,resample=TRUE),a,resample=FALSE)
b <- c('source'=a,'contract'=a1,'aggregation'=a2)
print(b)
display_brick(b,grid=TRUE
              ,grid.lon=(seq(ncol(a)*expand+1)-1)/expand
              ,grid.lat=(seq(nrow(a)*expand+1)-1)/expand)
session_grid(NULL)
a <- .makeRaster(6,8)
expand <- 3
b <- c("source"=regrid(a,mul=expand,resample=FALSE,resetGrid=FALSE)
      ,"simple"=regrid(a,mul=expand,cascade=TRUE,resetGrid=FALSE)
      ,"cascaded"=regrid(a,mul=expand,cascade=FALSE,resetGrid=FALSE))
print(b)
display_brick(b)
session_grid(a)
eps <- 1e-4
r <- c(0,expand^(-2)-eps,expand^(-2)+eps,1,expand^0.5
      ,(expand+2/3)^2-eps,(expand+2/3)^2+eps,99)
g2 <- regrid(mul=expand)
session_grid(g2)
b <- ursa_new(bandname=sprintf("Resample=%.4f",r))
for (i in seq(b))
  b[i] <- regrid(a,g2,resample=r[i])
print(b)
display_brick(b,layout=c(2,NA)
              ,grid=TRUE,grid.lon=seq(ncol(a)+1)-1,grid.lat=seq(nrow(a)+1)-1)

```

rep

Replicate bands of raster image.

Description

rep for object of class ursaRaster creates new ursaRaster objects with repetition of original band sequence.

Usage

```

## S3 method for class 'ursaRaster'
rep(x, ...)

```


Arguments

`x` Object of class `ursaRaster`
`...` Further arguments to be passed to or from other methods. Keywords:
 `times` Positive integer. Number of times to repeat each band.
 If argument has no name, then `times` is assumed.

Value

Object of class `ursaRaster`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[c](#) for `ursaRaster`.

Examples

```
session_grid(NULL)
session_grid(regrid(mul=1/4))
a <- ursa_dummy(nband=3)
print(a)
b1 <- rep(a,by=2)
print(b1)
b2 <- rep(a,length=5)
print(b2)
b3 <- rep(a[3],3)
print(b3)
```

Replace

assign values to the portion of raster images

Description

This operator is used to set or replace values in portion of bands or lines in `ursaRaster` object in memory or data writing to file.

Usage

```
## S3 replacement method for class 'ursaRaster'
x[i, j, ...] <- value
```

Arguments

x	ursaRaster object
i	Integer or character. If integer, then band index, specifying bands to replace. If character, either list of band names or character sting for regular expression to match band index. In the (<i>spatial</i> , <i>temporal</i>) interpretation of ursaRaster object j points to <i>temporal</i> component.
j	Mentioned for consistence with internal generic function [<code><-</code>].
...	Mentioned for consistence with internal generic function [<code><-</code>]. Use regexp=FALSE for matching by match , and regexp=TRUE for matching by Perl-compatible regexps case insensitive grep . Default is FALSE.
value	ursaRaster object or numeric (scalar, matrix , array). The latter is coerced to internal matrix of \$value item of ursaRaster object.

Details

Operator `\sQuote{[<-]}` is high-level implementation for data writing. If `x$value` item is not applicable, then value of ursaRaster is not in memory. In this case the controlled by `i` and `j` portion is written to file. If both `i` and `j` are missing, then `x[] <- value` writes values to file wholly.

It is not implemented the simultaneously writing to file portion of bands and portion of lines.

Files (currently, ENVI Binary) are opened for reading and writing.

Value

If values of ursaRaster object are in memory, then modified ursaRaster object with replaced bands or lines.

If values of ursaRaster object are not applicable, then ursaRaster object *as is*.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[Extract](#)

Examples

```
session_grid(NULL)
## Prepare
session_grid(regrid(mul=1/4))
a <- pixelsize()
w <- c("first", "second", "third", "fourth", "fifth", "sixth")
b1 <- rep(a/mean(a), length(w)) + seq(length(w)) - 1
bandname(b1) <- w
nr <- ursa_rows(b1)
bottom <- (as.integer(nr/2)):nr
write_envi(b1, "tmp1", compress=FALSE, interleave="bil")
b2 <- b1
```

```

print(b1)

## Replace
b2[1] <- 10+b1["second"]
b2[2] <- 20
try({
  data(volcano)
  b2[3] <- 30+volcano
}) ## error: unable to coerce
b2["fourth"] <- 40+as.matrix(b1[3])
b2[5] <- 50+as.array(b1[4])
set.seed(352)
try(b2["six"] <- 60+6+runif(5,min=-1,max=1)) ## Data structures mismatching
print(b2)
print(object.size(b2))

## Write
b3 <- create_envi(b2,"tmp2")
print(object.size(b3))
for (i in chunk_line(b3,0.04))
{
  b3[,i] <- b2[,i]+100
  if (5 %in% i)
    print(object.size(b3))
}
close(b3)
print(object.size(b3))
b4 <- read_envi("tmp2")
print(b4)
envi_remove("tmp[12]")

```

seq

Sequence Generation for raster image and coordinate grid

Description

Set of functions to generate regular sequences of bands, x-/y-coordinates and columns/rows.

Usage

```

## S3 method for class 'ursaRaster'
seq(...)

## S3 method for class 'ursaGrid'
seq(...)

ursa_seqx(obj)
ursa_seqy(obj)
ursa_seqc(obj)
ursa_seqr(obj)

```

Arguments

- ... Set of arguments, which are recognized via their names (using [regular expressions](#)), position and classes.
- . * First argument (position 1). Object of classes `ursaRaster`, `ursaGrid`.
 - . * Second argument (position >1). One-character name. Valid values are in the list `c("z", "x", "y", "c", "r", "lines", "samples")`. "c" ("samples") and "r" ("lines") specify to generate cell sequence in the horizontal and vertical directions from bottom-left corner, whereas "z" specifies to generate sequence of bands. x and "y" return cell midpoints in spatial dimension.
- obj Object of classes `ursaRaster`, `ursaGrid`. Missing obj is allowed; in this case the session grid is considered.

Details

All ordinal sequences (axis is `\dQuote{c}`, `\dQuote{r}`, `\dQuote{z}`) start from 1L.
 axis=`\dQuote{z}` is ignored in the function `seq` for `ursaGrid` object. The returned value is 1L.
`seq(obj)` for `ursaRaster` objects is suitable for using in cycles across bands.

Value

Functions `ursa_seqx` and `seq(obj, "x")` return x-coordinates of cell midpoints.
 Functions `ursa_seqy` and `seq(obj, "y")` return y-coordinates of cell midpoints.
 Functions `ursa_seqc`, `seq(obj, "samples")` and `seq(obj, "c")` return sequence of cells in horizontal direction.
 Functions `ursa_seqr`, `seq(obj, "lines")` and `seq(obj, "r")` return sequence of cells in vertical direction.
 Functions `seq(obj)` and `seq(obj, "z")` for `ursaRaster` object returns sequence of bands.
 Function `seq(obj)` and `seq(obj, "z")` for `ursaGrid` object returns 1L.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
session_grid(regrid(mul=1/16))
print(session_grid())
a <- ursa_dummy(nband=5)
print(a)
print(seq(a))
print(seq(a, "c"))
print(seq(a, "x"))
print(ursa_seqx())
```

session	<i>Get and set sessional parameters for grid, CRS, external software for open PNG files.</i>
---------	--

Description

`session_grid` without arguments returns current grid properties. `session_grid` with arguments specifies grid, which is used by functions of this package, e.g., for plotting, for opened and created raster images during current session.

`session_pngviwer` is used to permit external software to open PNG files.

`session_tempdir` specifies directory for temporal files in some cases.

`session_use_experimental_functions` allows to use undocumented (experimental) functions.

Group of functions `session_proj4`, `session_crs`, `session_cellsize`, `session_bbox` extracts certain properties of sessional grid.

Usage

```
session_grid(obj, ...)
session_proj4()
session_crs()
session_cellsize()
session_bbox()
session_dim()

session_pngviewer(allow = NA)
session_tempdir(dst = character())
session_use_experimental_functions()
```

Arguments

<code>obj</code>	Either missing, or NULL, or file name, or object of class <code>ursaRaster</code> , or object of class <code>ursaGrid</code> , or spatial object (simple features (sf), spatial abstracts (sp)).
<code>allow</code>	Logical. If TRUE then it is allowed to use external software for viewing PNG files. NA is interpreted as TRUE in the case of "Rscript" usage, and interpreted as FALSE in the case of interactive session or "R CMD BATCH" usage. Default is NA.
<code>dst</code>	Character. Directory name for temporal files. Empty character or non-character is interpreted as <code>getwd()</code> in the case of "Rscript" usage, and interpreted as <code>tempdir()</code> in the case of interactive session or "R CMD BATCH" usage. Default is <code>character()</code> (empty character).
<code>...</code>	Optional arguments passing to <code>regrid</code> at first.

Details

`session_grid` deals with option “`ursaSessionGrid`”: `options(ursaSessionGrid=...)` or `getOption("ursaSessionGrid")`

Usage `session_grid()` without arguments return value of “`ursaSessionGrid`” option via calling `getOption("ursaSessionGrid")`. If `is.null(getOption("ursaSessionGrid"))` then `session_grid()` returns default CRS.

Usage `session_grid(NULL)` resets “`ursaSessionGrid`” option via calling `options(ursaSessionGrid=NULL)`.

The sequential calling

```
session_grid(NULL)
session_grid()
```

returns default CRS. For checking that the option has been reset successfully, use `getOption("ursaSessionGrid")` after `session_grid(NULL)`

`session_proj4` and `session_crs` are synonyms.

Value

Object of class `ursaGrid`. It is a list. Default values are grid parameters of NSIDC polar stereo gridded data of Northern hemisphere with nominal gridded resolution 25 km. (<https://nsidc.org/data/user-resources/help-center/guide-nsidcs-polar-stereographic-projection>)

```
List of 9
 $ columns: int 304
 $ rows   : int 448
 $ resx   : num 25000
 $ resy   : num 25000
 $ minx   : num -3850000
 $ maxx   : num 3750000
 $ miny   : num -5350000
 $ maxy   : num 5850000
 $ proj4   : chr "+proj=stere +lat_0=90 +lat_ts=70.0 +lon_0=-45.0 +k=1
              +x_0=0.0 +y_0=0.0 +a=6378273.000 +b=6356889.449 +units=m +no_defs"
 - attr(*, "class")= chr "ursaGrid"
NULL
```

`session_proj4` and `session_crs` return item `proj4`.

`session_cellsize` returns squared root from multiplication of cell dimension: `sqrt(resx*resy)`.

`session_pngviewer` returns value of `getOption("ursaAllowPngViewer")`.

`session_bbox` returns named numeric of length 4: minimal x-coordinate (`xmin`), minimal y-coordinate (`ymin`), maximal x-coordinate (`xmax`), maximal y-coordinate (`ymax`).

`session_dim` returns named integer of length 2: number of rows (lines) and number of columns (samples).

`session_use_experimental_functions` added some non-public functions to current namespaces and returns invisible list of function names.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

Class [ursaGrid](#). Use [regrid](#) to partial grid changing.

Examples

```
session_grid(NULL)
getOption("ursaSessionGrid") ## NULL
(g1 <- session_grid()) ## default
g1$resx <- g1$resy <- 12500
g1$columns <- as.integer(with(g1,(maxx-minx)/resx))
g1$rows <- as.integer(with(g1,(maxy-miny)/resy))
session_grid(g1)
session_grid(NULL)
a <- ursa_new(value=3)
session_grid(a)

print(session_pngviewer())
```

 sort

Sort multiband raster by band names.

Description

Changing order of bands based on sorting of band names.

Usage

```
## S3 method for class 'ursaRaster'
sort(x, decreasing = FALSE, ...)
```

Arguments

x	Object of class ursaRaster
decreasing	Logical. Should the sort be increasing or decreasing? Not available for partial sorting. Default is FALSE.
...	Other arguments, which are passed to S3 method for sorting characters.

Details

Function `sort()` for ursaRaster assumes bands reordering based on character band names.

Value

Object of class ursaRaster

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
a <- ursa_dummy(nband=7L)
a
sort(a)
sort(a,decreasing=TRUE)
```

spatial_engine	<i>Wrapper functions for manipulation with non-raster spatial objects</i>
----------------	---

Description

These wrappers return iniform properties or do consimilar manipulations for spatial objects of different types: simple features (package **sf**) and abstract class Spatial (package **sp**). Appropriate functionality (“*engine*”) of respective packages is used.

Usage

```
spatial_engine(obj, verbose = FALSE)

spatial_crs(obj, verbose = FALSE)
spatial_proj(obj, verbose = FALSE)
spatial_proj4(obj, verbose = FALSE)

spatial_crs(obj, verbose = FALSE) <- value
spatial_proj(obj, verbose = FALSE) <- value
spatial_proj4(obj, verbose = FALSE) <- value

spatial_bbox(obj, verbose = FALSE)
spatial_bbox(obj, verbose = FALSE) <- value

spatial_data(obj, subset= ".+", drop = NA, verbose = FALSE)
spatial_data(obj, verbose = FALSE) <- value

spatial_geometry(obj, verbose = FALSE)
spatial_geometry(obj, verbose = FALSE) <- value

spatial_geotype(obj, each = FALSE, verbose = FALSE)
spatial_shape(obj, each = FALSE, verbose = FALSE)

spatial_transform(obj, crs, verbose = FALSE, ...)

spatial_coordinates(obj, verbose = FALSE)
spatial_centroid(obj, verbose = FALSE)
```



```
spatial_fields(obj, verbose = FALSE)
spatial_colnames(obj, verbose = FALSE)

spatial_fields(obj, verbose = FALSE) <- value
spatial_colnames(obj, verbose = FALSE) <- value

spatial_area(obj, verbose = FALSE)

spatial_dim(obj, verbose = FALSE)

spatial_count(obj, verbose = FALSE)

spatial_nrow(obj, verbose = FALSE)
spatial_ncol(obj, verbose = FALSE)

spatial_filelist(path = ".", pattern = NA, full.names = TRUE, recursive = FALSE,
                  ignore.case = TRUE)
spatial_dir(path = ".", pattern = NA, full.names = TRUE, recursive = FALSE,
             ignore.case = TRUE)

spatial_basename(fname)
spatial_pattern(fname)

is_spatial(obj, verbose = FALSE)

is_spatial_points(obj, verbose = FALSE)
is_spatial_lines(obj, verbose = FALSE)
is_spatial_polygons(obj, verbose = FALSE)

spatial_intersection(x, y,
                    geometry=c("default", "polygons", "lines", "points", "all"),
                    verbose = FALSE)
spatial_symdifference(x, y, verbose = FALSE)
spatial_difference(x, y, verbose = FALSE)
spatial_union(x, y, byid=NA, verbose = FALSE)
spatial_crop(x, y)

spatial_buffer(obj, dist = 0, quadsegs = 30L, verbose = FALSE)

spatial_trim(obj)

spatial_valid(obj, each = FALSE, reason = FALSE, verbose = FALSE)

spatial_grid(obj)

spatial_bind(...)
```

Arguments

obj	Simple feature (package sf) or Spatial abstract class (package sp) for all functions, excepting <code>spatial_geometry<-</code> . Data frame for <i>Replace</i> function <code>spatial_geometry<-</code> .
x, y	Objects of simple feature (package sf) class or Spatial abstract class (package sp). Spatial abstracts are not applicable for <code>spatial_crop()</code> .
crs	Projection EPSG code or projection PROJ.4 string.
subset	Pattern to field names (colnames) of attribute table (data frame) for subsetting using regular expressions . By default, all fields are selected.
drop	Logical. Dropping column of data frame. If TRUE, then vector of data is returned. If FALSE, then structure of data is kept. Default is NA, which is interpreted as TRUE for single column and as FALSE for multiple columns.
value	Value for property assignment in <i>replacement</i> functions. Either numeric EPSG code or character PROJ.4 string for <code>spatial_crs<-</code> and <code>spatial_proj4<-</code> . Spatial object or geometry of spatial object for <code>spatial_geometry<-</code> .
path	See description of argument <code>path</code> in function dir .
pattern	See description of argument <code>pattern</code> in function dir .
full.names	See description of argument <code>full.names</code> in function dir .
recursive	See description of argument <code>recursive</code> in function dir .
ignore.case	See description of argument <code>ignore.case</code> in function dir .
quadsegs	Integer. Number of segments per quadrant (fourth of a circle), for all or per-feature. See description for <code>nQuadSegs</code> argument of st_buffer .
dist	Numeric. Buffer distance for all, or for each of the elements. See description for <code>dist</code> argument of st_buffer .
byid	Logical. For <code>spatial_union</code> function, TRUE does unite of each feature; FALSE returns a single feature that is the geometric union of the set of features; default NA is coerced to FALSE for unary operation (missing y) and to TRUE for binary operation.
fname	Character. Filename (source or packed) of spatial data.
each	Logical. Whether result will be returned for each record (TRUE) or generalized (FALSE). Default is FALSE.
geometry	Character. Desired output geometry for <code>engine="sf"</code> . If "default" then output geometry is defined internally (e.g., "polygons" for polygons intersection). If "all" then no output subsetting. Default is "default".
reason	Logical. If TRUE, then the reason for validity ("Valid Geometry") or invalidity is returned. If FALSE, then logical value of validity is returned. Default is FALSE.
verbose	Logical. Value TRUE provides information on console. Default is FALSE.
...	1) Spatial objects for function <code>spatial_bind</code> ; 2) Further arguments in function <code>spatial_transform</code> passed to <code>sf::st_transform</code> or to <code>sp::spTransform</code> .

Value

`spatial_engine` returns package name (character string "sf" or "sp"), which functionality is used for manipulation with spatial object `obj`.

`spatial_crs` and `spatial_proj4` are synonyms, The *Extract* functions return projection string in the PROJ.4 notation; the *Replace* functions change projection property of the object.

`spatial_bbox` (*Extract* function) returns numeric vector of length 4 with names "xmin", "ymin", "xmax" and "ymax".

`spatial_bbox<-` (*Replace* function) assigns boundary bbox to the object; it is valid only for objects of Spatial abstract class (package **sp**).

`spatial_data` (*Extract* function) returns attribute table only, without geometry. Subsetting fields can be specified by argument `subset` using regular expressions. If `drop=TRUE` and selected single column then vector is returned instead of data frame.

`spatial_data<-` (*Replace* function) adds spatial data to the object geometry. Source data (if presents) are dropped.

`spatial_geometry` (*Extract* function) returns only geometry, which format is depended on class of `obj`.

`spatial_geometry<-` (*Replace* function) adds geometry to the object.

`spatial_transform` does a transformation of spatial coordinates to the new CRS and returns object of the same class as class of `obj`.

`spatial_geotype` and `spatial_shape` are synonyms; each returns type of spatial data: "POINT", "LINESTRING", "POLYGON", "MULTIPOLYGON",

`spatial_coordinates` returns simplified matrix or list of coordinates of original object.

Extract functions `spatial_fields` and `spatial_columns` return column names of spatial attributive table. `spatial_columns` is synonym to `spatial_fields`.

Replace functions `spatial_fields<-` and `spatial_columns<-` change column names of spatial attributive table. `spatial_columns<-` is synonym to `spatial_fields<-`.

`spatial_area` is valid for polygonal geometry. It returns area of polygons.

`spatial_length` is valid for linear geometry. It returns length of lines.

`spatial_dim` gets dimension of spatial coordinates; it returns either 2L (XY) or 3L (XYZ).

`spatial_count` returns number of items of object geometry.

`spatial_nrow` and `spatial_ncol` return number of rows and number of columns of attributive table.

`spatial_filelist` and its synonym `spatial_dir` return list of files with file extensions, which are associated with certain GIS vector formats. The function's basis is [dir](#).

`spatial_basename` returns basename (without extension) of file `fname` of spatial object.

`spatial_pattern` returns pattern of `spatial_basename` for using in [regular expressions](#).

`is_spatial` returns logical value does the object belong to the class of spatial data.

`is_spatial_points` returns logical value does the object have point geometry.

`is_spatial_lines` returns logical value does the object have (multi)linestring geometry.

`is_spatial_polygons` returns logical value does the object have (multi)polygonal geometry.

`spatial_intersection` returns intersection of two spatial objects.
`spatial_difference` returns difference of two spatial objects.
`spatial_symdifference` returns symmetric difference of two spatial objects.
`spatial_buffer` returns buffered spatial object.
`spatial_union` returns combined geometry without internal boundaries.
`spatial_crop` returns cropped geometry of first spatial object by second spatial object of boundary box derived from spatial object.
`spatial_trim` returns spatial object without extra attributes added by **ursa** package.
`spatial_grid` generates suitable spatial grid from input vector and returns object of class `ursaGrid`.
`spatial_centroid` returns spatial centroid.
`spatial_bind` returns spatial object concatenated from input spatial objects.

Acknowledgements

The great improvement for development of functions for manipulation with spatial objects has been reached during work in series of projects (2015-2018) for design of marine protected areas in the Arctic.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

References

Classes and methods in packages **sf** and **sp** help.

Examples

```

session_grid(NULL)
n <- 1e2
x <- runif(n,min=25,max=65)
y <- runif(n,min=55,max=65)
z <- runif(n,min=1,max=10)
da <- data.frame(x=x,y=y,z=z)
if (requireNamespace("sp")) {
  da.sp <- da
  sp::coordinates(da.sp) <- ~x+y
  sp::proj4string(da.sp) <- "+init=epsg:4326"
  print(spatial_bbox(da.sp))
  print(spatial_crs(da.sp))
}
if (requireNamespace("sf")) {
  da.sf <- sf::st_as_sf(da,coords=c("x","y"),crs=4326)
  print(spatial_bbox(da.sf))
  print(spatial_crs(da.sf))
}

```

spatial_levelsplit	<i>Drops spatial object with overlapped geometry to spatial object with non-overlapped geometry.</i>
--------------------	--

Description

Contour (levels after kernel utilization distribution, isochrones and other isolines) polygonizations produces set of polygon layers, which geometry is overlapped. Plot of such polygons may cause the invisibility the less polygon behind the larger polygon. This function makes consequent geometries. Simplified, concentrated circles are dropped to non-overlapped rings.

Usage

```
spatial_levelsplit(obj, sep = " - ")
```

Arguments

obj	Spatial object, either simple features (package sf) or abstract class Spatial (package sp)
sep	Separator between concatenation of two values

Value

Spatial object, which class is the same as class of obj.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
palette("Set3")
radius <- seq(1,length.out=5,by=1)*200
ct <- ursa_colortable(colorize(radius,alpha=0.5,pal=sample(palette()),length(radius)))
origin <- sf::st_sfc(sf::st_point(c(lon=139.2,lat=36.6)),crs=4326)
origin <- spatial_transform(origin,"EPSG:6671")
isopoly <- do.call(spatial_bind,lapply(radius*1e3,function(r) spatial_buffer(origin,r)))
spatial_data(isopoly) <- data.frame(radius=radius)
isointerval <- spatial_levelsplit(isopoly)
isointerval$radius
ct2 <- ursa_colortable(colorize(isointerval$radius,pal=unname(ct)))
session_grid(isopoly,border=20)
compose_open(2,legend=list("left","right"))
compose_panel(isopoly,col=ct
              ,annotation.text="Semi-transparent colors are overlapped")
compose_panel(isointerval,col=ct2
              ,annotation.text="Not overlapped rings")
compose_legend(list(ct,ct2))
compose_close()
```

`spatial_read`*Wrapper functions for reading spatial objects.*

Description

Read either simple features (package **sf**) from disk using appropriate functionality (“*engine*”) of respective packages is used.

Usage

```
spatial_read(dsn, engine = c("native", "sf"))
```

Arguments

<code>dsn</code>	Character. File name of spatial object (vector GIS).
<code>engine</code>	Character. Functionality of which package is used for reading data. If value is “sf”, then package sf is used and simple features are returned. If value is “geojsonsf”, GDAL driver is GeoJSON and package geojsonsf can be loaded, then package geojsonsf is used and simple features are returned. If value is “sp” and package sp can be loaded, then Spatial abstracts (package sp) are returned. If value is “native” then engine selection depends on has suggested package sf been installed and is there possibility to use geosonjf for GeoJSON driver.

Details

Currently, list of arguments of this funtion is simplified and can be expanded.

Value

Depending of used engine, either simple features (package **sf**) or Spatial abstracts (**sp**).

Note

For GeoJSON files in the case `engine="geojsonsf"` reading is faster and the order of fields can be rearranged.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[read_sf](#)
[spatial_write](#)

Examples

```
session_grid(NULL)
n <- 1e2
x <- runif(n,min=25,max=65)
y <- runif(n,min=55,max=65)
z <- runif(n,min=1,max=10)
da <- data.frame(x=x,y=y,z=z)
if (requireNamespace("sf",quietly=TRUE)) {
  obj1 <- sf::st_as_sf(da,coords=c("x","y"),crs=4326)
  print(series(obj1))
  fname1 <- file.path(tempdir(),"res1.shp")
  print(fname1)
  spatial_write(obj1,fname1)
  res1 <- spatial_read(fname1,engine="sf")
  print(series(res1))
}
print(spatial_dir(tempdir()))
```

spatial_write

Wrapper functions for writing spatial objects.

Description

Write spatial object to disk. If spatial object is Simple Features, then appropriate functions from package **sf** are used. If spatial object are abstract of class Spatial (package **sp**) then preliminary transformation to Simple Features is performed.

Usage

```
spatial_write(obj, fname, layer, driver = NA, compress = "",
             dopt = character(), lopt = character(), verbose = FALSE, ...)
```

Arguments

obj	Spatial object: Either Simple Features (sf) or Spatial Abstract (sp). List of spatial objects can be used.
fname	Character. File name with or without extension. If extension is missed, then argument driver must be specified.
layer	Character. Layer name. If missed, then basename of fname is used.
driver	Character. Driver for specification of output file format. Default is NA; value is determined from extension of fname.
compress	Character or logical. Will output file or list of files be packed after writing and what archive format will be used. Available character values are "" (default; no compression), "gz", "gzip", "bz2", "bzip2", "zip", "xz", "zstd", "zst". If logical and TRUE, then "zip" is used for driver "ESRI Shapefile" and "gzip" otherwise. If logical and FALSE, then no compression.

dopt	Named character or named list. Driver dependent dataset creation options, , similar to argument dataset_options in function <code>st_write()</code> .
lopt	Named character or named list. Driver dependent layer creation options, similar to argument layer_options in function <code>st_write()</code> .
verbose	Logical. Value TRUE provides information on console. Default is FALSE.
...	Argument, which are passed to functions from suggested packages, e.g. pretty = TRUE in the case of using package yyjson .

Details

Based on `sf::st_write` function with additional options: compressing of output file(s), coordinates transforming (to longitudes and latitudes for driver="GeoJSON"), creating multi-layer destination (for driver="SQLite").

Value

invisible NULL.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

`write_sf`
`spatial_read`

Examples

```
session_grid(NULL)
n <- 1e2
x <- runif(n,min=25,max=65)
y <- runif(n,min=55,max=65)
z <- runif(n,min=1,max=10)
da <- data.frame(x=x,y=y,z=z)
if (requireNamespace("sf",quietly=TRUE)) {
  obj1 <- sf::st_as_sf(da,coords=c("x","y"),crs=4326)
  print(series(obj1))
  fname1 <- file.path(tempdir(),"res1.shp")
  print(fname1)
  spatial_write(obj1,fname1)
  res1 <- spatial_read(fname1,engine="sf")
  print(series(res1))
}
print(spatial_dir(tempdir()))
```

summary

Summary of raster image.

Description

Function summary for ursaRaster object produces summaries for each band.
Function summary for ursaValue object produces summaries for all values of raster image regardless of bands.

Usage

```
## S3 method for class 'ursaRaster'  
summary(object, ...)  
  
## S3 method for class 'ursaNumeric'  
summary(object, ...)  
  
## S3 method for class 'ursaCategory'  
summary(object, ...)
```

Arguments

object	Object of classes ursaRaster, ursaNumeric, or ursaCategory
...	Additional arguments affecting the summary produced.

Details

summary for ursaRaster object applies [summary](#) to each column of two-dimensions value matrix and collating the results.
summary for ursaValue object drops dimensions and applies [summary](#) to a vector.

Value

summary for ursaRaster object returns value of function [summary.matrix](#).
summary for ursaValue object returns object of class `"summaryDefault"`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[summary](#) in package **base**.

Examples

```
session_grid(NULL)
session_grid(regrid(mul=1/4))
a <- ursa_dummy(nband=3)
print(summary(a))
print(summary(ursa_value(a)))
print(a)
```

temporal_interpolate *Fill gaps across bands using moving mean window*

Description

temporal_interpolate is applicable for multiband raster image, where bands are regular times-tamps or period. For each cell (*local* operation of map algebra), NA value is replaced by averaging of two closest values (one value before, one value later) inside of moving window.

Usage

```
temporal_interpolate(obj, win = 7, cover = 0, verbose = FALSE)
```

Arguments

obj	Object of class <code>ursaRaster</code> or <code>matrix</code> , where spatial locations are by rows and temporal observations are by columns.
win	Positive integer. Size of moving window. Required odd value; otherwise is coerced to the closest odd integer.
cover	Not applicable. For consistence call with <code>temporal_mean</code> .
verbose	Logical. TRUE provides some additional information on console. Default is FALSE.

Details

Function uses weighted averaging depending of proximity of found non-NA values. For example, if `ind` is temporal index of NA value in the center of moving window, `indL=ind-2` is temporal index of the closest early value `valL`, and `indR=ind+1` is temporal index of the closest late value `valR`, then result is `val <- (1/3) * valL + (2/3) * valR`.

Value

`ursaRaster` object, if `obj` is object of class `ursaRaster`.
`matrix` object, if `obj` is a `matrix`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[temporal_mean](#)

Examples

```
session_grid(NULL)
n <- 45 # bands
m <- 3 # sample size
k <- median(seq(n))+seq(m)-(m %% 2)-1 ## sample subset
s <- 5 # window size
a <- round(ursa_dummy(n,min=-60,max=60,elements=15,mul=1/8))
a[a<(-40)] <- NA
b <- temporal_interpolate(a,7)
p1 <- colorize(a,lazy=TRUE)
p2 <- colorize(b,lazy=TRUE,colortable=p1)
display(list('Source'=p1[k], 'Gaps are filled'=p2[k]),layout=c(2,NA)
        ,legend=list(list(1,"right"),list(2,"right")),decor=FALSE)
```

temporal_mean	<i>Smooth value across bands using moving mean window</i>
---------------	---

Description

temporal_mean is applicable for multiband raster image, where bands are regular timestamps or period. For each cell (*local* operation of map algebra), the values are averaged using moving window.

Usage

```
temporal_mean(obj, win = 7, cover = 0, verbose = FALSE)
```

Arguments

obj	Object of class ursaRaster or matrix, where spatial locations are by rows and temporal observations are by columns.
win	Positive integer. Size of moving window. Required odd value; otherwise is coerced to the closest odd integer.
cover	Numeric in the interval $0 \leq \text{cover} \leq 1$ or positive numeric >1 . The required amount of non-NA elements in window to do a filtering. Otherwise, NA value is in output cell. If $\text{cover} \leq 1$ then amount is relative to window size. Default is 0: NA values are produced only if all elements in window have NA value.
verbose	Logical. TRUE provides some additional information on console. Default is FALSE.

Details

temporal_mean is similar to function `runmean(x=obj, k=win, endrule="mean")` from package **caTools**.

Value

ursaRaster object, if obj is object of class ursaRaster.

matrix object, if obj is a matrix.

Advanced

temporal_mean is only smoothing of time-series. For time-series analysis and processing it is suggested to apply lower-level approach.

`as.matrix` (for ursaRaster object with argument `coords=FALSE`) or `ursa_value` return matrix with spatial component by rows and temporal component by columns. It is possible to use `apply` with argument `MARGIN=1` to this matrix. If `apply` returns matrix Y, then this matrix can be coerced to ursaRaster object by calling `as.ursa` with argument `t(Y)`.

```
X <- as.matrix(obj)
Y <- apply(X, 1, function(x) {y <- do_something_return_matrix(x); y})
res <- as.ursa(t(Y))
```

For example, package **caTools** provides some functions for manipulation with moving window.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

References

Package **caTools** <https://CRAN.R-project.org/package=caTools>

See Also

`caTools::runmean` (click if package **caTools** is installed)

Examples

```
session_grid(NULL)
set.seed(352)
n <- 45 # bands
m <- 3 # sample size
k <- median(seq(n))+seq(m)-(m %% 2)-1 ## sample subset
s <- 5 # window size
a <- round(ursa_dummy(n,min=-60,max=60,elements=15,mul=1/8))

## namespace of package 'caTools' is required
if (requireNamespace("caTools")) {
  b1 <- as.ursa(t(apply(as.matrix(a),1,caTools::runmean,k=s,endrule="mean")))
  b2 <- temporal_mean(a,s)
  print(b1[k])
  print(b2[k])
  print(c('identical?'=all.equal(ursa_value(b1),ursa_value(b2))))
}
```

```

a[a<(-40)] <- NA
va <- as.matrix(a) # or 'ursa_value(a)'
b3 <- temporal_mean(a,s,cover=3/4,verbose=TRUE)
b4 <- as.ursa(temporal_mean(as.matrix(va),s,cover=3/4,verbose=TRUE))
p <- list('Before moving window'=a[k]
        , 'After moving window'=b3[k]
        , '\temporal_mean\' to matrix'=b4[k])
print(p)
print(c('identical?'=all.equal(ursa_value(b3),ursa_value(b4))))
display(p[1:2],legend=list(list(1,"right"),list(2,"right")),decor=FALSE)

```

trackline

*Create segmented line from points' sequence***Description**

Connect sequence of points (locations) by direct lines (tracks)

Usage

```
trackline(obj, by=NULL, connect=c("united", "consequent"), gentle=FALSE)
```

Arguments

obj	Simple feature (package sf) or Spatial abstract class (package sp) with POINTS spatial geometry.
by	Either field name or NULL. If specified, then value "united" is applied for argument connect, returned spatial object is splitted regarding values of by with linked single-column attribute table with specified field. Default is NULL.
connect	Structure of output segments; either sequence of single segments ("consequent") or single multi-segment ("united").
gentle	Logical. Value TRUE directs repetition of binded columns of data table. Value FALSE omits duplicated column values. Default is FALSE.

Details

Function generates n-1 segments from n input points. Data (attribute table) is trasfered to output object with excluding of first row.

gentle=TRUE may be useful to keep heterogenic data structure for spatial binding ([spatial_bind](#)).

Value

Simple feature (package **sf**) or Spatial abstract class (package **sp**) with LINESTRING spatial geometry.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
n <- 15
lon <- rev(sort(runif(n,min=40,max=60)))
lat <- sort(runif(n,min=30,max=50))
pt <- data.frame(lon=lon,lat=lat,value=seq(n))
if (requireNamespace("sp")) {
  sp::coordinates(pt) <- c("lon", "lat")
  sp::proj4string(pt) <- "EPSG:4326"
} else {
  pt <- sf::st_as_sf(pt, coords=c("lon", "lat"), crs="WGS84")
}
ct <- ursa_colortable(colorize(pt$value))
tr <- trackline(pt, connect="consequent")
#opW <- options(warn=0)
session_grid(pt, expand=1.1)
compose_open(2)
panel_new()
panel_plot(pt, col=ct)
panel_decor()
panel_new()
panel_plot(tr, col=ct, lwd=3)
panel_decor()
compose_legend(ct, unit="step number")
compose_close()
```

ursa

Get and set properties of raster image.

Description

For package description see.

ursa is a wrapper to initialize object of class ursaRaster, to get and to set properties for this object.

Usage

```
ursa(obj, attr, ...)
ursa(obj, attr, ...) <- value
```

Arguments

obj	Any numeric structure (scalar, matrix, array) for initializing. Object of class ursaRaster for extracting and assigning properties.
attr	Character. Name of property.

... Arguments, which are passed for properties specification.
 value Value for property assignment.

Details

Initializing function ursa with missing argument attr is a wrapper for function as.ursa.

<i>Matched pattern</i>	<i>Replace method?</i>	<i>Description of property</i>	<i>Implementation</i>
"grid"	Yes	Raster grid (extent, projection, cellsize)	ursa_grid
"(proj crs)"	Yes	Coordinate reference system	ursa_proj
"val"	Yes	Raster value in the internal storage format	ursa_value
"(colort ct)"	Yes	Color table	ursa_colortable
"(categ class)"	Yes	Names of categories	names(ursa_colortable(
"name"	Yes	Band names	names
"(nodata ignorevalue bg)"	Yes	Value, which is interpreted as NA.	ignorevalue
"^table\$"	No	Frequency of unique values	as.table
"cell"	No	<i>Squared</i> cell size	with(ursa_grid(obj),sq
"^dim\$"	No	Dimension of raster image	dim
"(extent bbox)"	No	Spatial extent of raster image	with(ursa_grid(obj),c(
"(nrow rows lines)"	No	Number of rows of raster image	ursa_grid(obj)\$rows
"(ncol columns samples)"	No	Number of columns of raster image	ursa_grid(obj)\$columns
"con"	No	structure of connection	obj\$con
"(info meta(data)*)"	No	Metadata, brief info	ursa_info
"^file(name)*"	No	Connection name (filename)	obj\$con\$fname

Argument ... is used to specify band index or band pattern in ursa(obj, "value", ...)

Value

Inititalizing function ursa (missing attr) returns object of class ursaRaster.

Extract function ursa returns object of respective property.

Replace function ursa<- returns object

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[as.ursa](#)

Examples

```
a1 <- ursa(volcano)
print(a1)
## to avoid over-timing during tests -- begin
  display(a1)
## to avoid over-timing during tests -- end
```

```

a2 <- ursa(volcano,flip=TRUE)
print(a2)
## to avoid over-timing during tests -- begin
  display(a2)
## to avoid over-timing during tests -- end

a3 <- ursa(volcano,permute=TRUE)
print(a3)
## to avoid over-timing during tests -- begin
  display(a3)
## to avoid over-timing during tests -- end

a4 <- ursa(volcano,flip=TRUE,permute=TRUE)
print(a4)
## to avoid over-timing during tests -- begin
  display(a4)
## to avoid over-timing during tests -- end

dima <- c(200,300,4)
b1 <- ursa(array(runif(prod(dima)),dim=dima))
print(b1)
display_brk(b1,scale=1,pal.rotate=0,pal.hue=0,decor=FALSE)
session_grid(NULL)

c1 <- ursa(seq(3))
print(c1)
c2 <- ursa(seq(3),bands=3)
print(c2)

c3 <- ursa(value=FALSE)
str(ursa(c3,"value"))

c4 <- ursa(bands=2,nodata=-99L)
print(c4)
print(ursa(c4,"nodata"))

c5 <- ursa(bandname=format(Sys.Date()+seq(7)-1,"%A"))
ursa(c5,"value") <- rev(seq(nband(c5)))
c5 <- colorize(c5)
ct <- ursa(c5,"colortable")
print(c5)

v <- ursa(c5[3:5],"value")
str(v)
v <- c(v)
str(v)
c6 <- ursa(v,colortable=ct)
print(c6)
print(ursa(c6,"colortable"))

```


Description

Class ursaConnection is a part of class ursaRaster. It defines storage of raster images and manipulations with reading and writing.

Usage

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

## S3 method for class 'ursaConnection'
seek(con, where = NA, origin = "start", rw = "", ...)
```

Arguments

x	ursaConnection object in function print.
con	ursaConnection object in function seek.
where	Passed to seek for class connection.
origin	Passed to seek for class connection.
rw	Passed to seek for class connection.
...	print: Further arguments passed to generic functions print and str . seek: Further arguments passed to function seek for class connection.

Details

ursaConnection get item \$con from ursaRaster object.

Functions print and is.con are for developers rather than users.

Non-public function .con.skeleton() is used to generate the blank ursaConnection object. This approach provides unified sequence of list's items:

Value

ursaConnection is a list. The most of names have a relation to specification of [ENVI Header Files](#). Items:

driver	Character. Keyword of supported image formats. Allowed "ENVI" or "GDAL".
samples	Integer. Number of image columns (samples)
lines	Integer. Number of image rows (lines)
bands	Integer. Number of image bands (channels, layers)
datatype	Integer. Keyword for data type (4 - 32-bit floating point, 2 - 16-bit signed integer, <i>etc</i>)
interleave	Character "bsq", "bil", "bip". Data interleave - streams of bytes in storage.
byteorder	Integer, 0 or 1. The order of bytes. byteorder=0 less significant byte first, byteorder=1 most significant byte first
endian	Character. See <code>\dQuote{endian}</code> argument in readBin and writeBin

swap	Integer 0 or 1. Passed to C-functions fread and fwrite
signed	Logical. Derived from \$datatype
offset	Integer. Header offset in binary file. Default is 0.
wkt	Logical. In ENVI Header files wkt=TRUE forced to use “coordinate system string” field instead of “projection info” field
nodata	Numeric. Replacement NA values in the storage.
mode	Character. storage.mode of data
sizeof	Integer, positive. Size in bytes for stored values. Extracted from \$datatype
indexC	Integer vector. Sample indices in spatial cropping.
indexR	Integer vector. Line indices in spatial cropping.
indexZ	Integer vector. Band indices
posC	Integer vector. Sample indices in partial reading.
posR	Integer vector. Line indices in partial reading.
posZ	Integer vector. Band indices in spatial cropping or partial reading.
fname	Character. File name. If driver=ENVI, then full path for ENVI binary file.
connection	Character. See connections
compress	Signed integer. 0L no compressing, -1L compressed file plus decompressed file, -2L decompressed file, 1L - file will be compressed.
seek	Logical. Does connection support seek ?
handle	connections in fact

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[ursa](#)(obj, "con")

Examples

```
session_grid(NULL)
print(methods(class="ursaConnection"))

a <- pixelsize()
write_envi(rep(a,5),"tmp1",compress=FALSE)
## change spatial domain for cropping example
g <- session_grid(regrid(lim=c(-1200000,-1400000,1600000,1800000)))
print(g)
b <- open_envi("tmp1")
d <- b[,30:70]
print(ursa(d[2:3],"con"))
close(b)
envi_remove("tmp1")
```

ursaCRS*Coordinate Reference System (CRS) for raster images.*

Description

Class ursaCRS is a part of class ursaGrid. It defines map projection.

Usage

```
## S3 method for class 'ursaCRS'
print(x, ...)
## S3 method for class 'ursaCRS'
str(object, ...)
```

Arguments

x	ursaCRS object in function print.
object	ursaCRS object in function str.
...	Further arguments passed to generic functions print , and str .

Value

Functions print information about CRS and return invisible NULL value.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- ursa_dummy()
crs <- ursa_crs(a)
print(c('Is proj4string used?'=p4 <- isTRUE(getOption("ursaProj4Legacy"))))
print(crs)
str(crs)
op <- options(ursaProj4Legacy=!p4)
print(c('Is proj4string used?'=p4 <- isTRUE(getOption("ursaProj4Legacy"))))
session_grid(NULL)
a <- ursa_dummy()
crs <- ursa_crs(a)
print(crs)
str(crs)
options(op)
```

ursaGrid	<i>Spatial parameters of raster images.</i>
----------	---

Description

Class ursaGrid is a part of class ursaRaster. It defines spatial locations of image.

Usage

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

## S3 method for class 'ursaGrid'
str(object, ...)

## S3 method for class 'ursaGrid'
dim(x)

## S3 method for class 'ursaGrid'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	ursaGrid object in functions print, dim and as.data.frame.
object	ursaGrid object in function str.
row.names	Ignored. Argument, which is passed to generic function as.data.frame .
optional	Ignored. Argument, which is passed to generic function as.data.frame .
...	Further arguments passed to generic functions as.data.frame , print , and str .

Details

The blank ursaGrid object is generated by calling of ursa_grid() without arguments. These approaches provide unified sequence of list's items:

```
List of 9
 $ columns: int NA
 $ rows   : int NA
 $ resx   : num NA
 $ resy   : num NA
 $ minx   : num NA
 $ maxx   : num NA
 $ miny   : num NA
 $ maxy   : num NA
 $ proj4   : chr ""
 - attr(*, "class")= chr "ursaGrid"
NULL
```

Value

Object of class ursaGrid is a list with items:

columns	Number of columns (samples)
rows	Number of rows (lines)
resx	Grid cell size by horizontal axis
resy	Grid cell size by vertical axis
minx	Left margin of boundary box
maxx	Right margin of boundary box
miny	Bottom margin of boundary box
maxy	Top margin of boundary box
proj4	PROJ.4 string

Function dim for object of class ursaGrid returns named vector of length 2: number of rows ("lines") and number of elements in a row ("samples")

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[regrid](#), [session_grid](#)

Examples

```
session_grid(NULL)
print(methods(class="ursaGrid"))

a <- pixelsize()
g <- ursa_grid(a)
print(is.ursa(a,"grid"))
print(is.ursa(g,"grid"))
print(g)
```

ursaProgressBar

Progress bar

Description

Informative progress bars with displaying elapsed and remained time.

Usage

```

ursaProgressBar(kind = c("tk", "txt"), title = .argv0(),
               label = "", min = 0, max = 1, initial = min, width = NA,
               style = 1, tail = FALSE, silent = FALSE)

setUrsaProgressBar(pb, value, title = NULL, label = NULL)

## S3 method for class 'ursaProgressBar'
close(con, ...)

```

Arguments

kind	Character. Type of progress bar. Valid values are "tk" to display progress bar in tcl/tk window and "txt" to display progress bar in terminal mode. Default is "tk".
style	See description for the same argument in txtProgressBar .
width	See description for the same argument in tkProgressBar .
title, label, min, max, initial, value, pb	See description for the same arguments in tkProgressBar and txtProgressBar .
con, ...	See description for the same arguments in close .
tail	Logical. Behaviour of progress bar appearing. If TRUE then progress bar will be used after progress step (e.g., at the end of routine). Default is FALSE (before progress step).
silent	Logical. If TRUE then progress bar will not appear; it can be useful for conditional scripting. Default is FALSE.

Details

Wrapper to one of [txtProgressBar](#), [tkProgressBar](#).

Visualization of progress bar is updated every 0.5 seconds, it is effective for multiple short-term iterations.

Progress bars should be closed by calling of appropriate method of generic function [close](#) depending on the class of reference progress bar.

Value

ursaProgressBar returns object of reference progress bar.

Note

Function name in style *camelCase* for consistency with other progress bar functions in R.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[txtProgressBar](#) [tkProgressBar](#)

Examples

```
session_grid(NULL)
n1 <- 3
n2 <- 83
p <- 0.0011
#require(tcltk)
pb <- ursaProgressBar(min=0,max=n1,title="first",tail=TRUE)
for (i in seq(n1)) {
  pb2 <- ursaProgressBar(min=0,max=n2,title="second")
  for (i in seq(n2)) {
    setUrsaProgressBar(pb2)
    Sys.sleep(p)
  }
  close(pb2)
  setUrsaProgressBar(pb)
}
close(pb)
```

ursaRaster

Definition of ursaRaster class.

Description

ursaRaster is S3 class for manipulation with georeferenced raster images. See ‘Value’ section.
is.ursa checks inhering to class ursaRaster

Usage

```
## S3 method for class 'ursaRaster'
print(x, digits = NA, grid = FALSE, raw = FALSE, caption = FALSE, ...)

## S3 method for class 'ursaRaster'
str(object,...)

is.ursa(obj, ref = NULL)
is_ursa(obj, ref = NULL)
```

Arguments

x, object	ursaRaster object.
obj	Any.
digits	Passed to format function

grid	Logical. If grid=TRUE then returns simplified metadata.
raw	Logical. If FALSE and values are categories, then attempting to restore numeric values from categorical names is before calculating of statistics. If TRUE then values for statistics are used as is. Default is FALSE.
caption	Logical of character. Print title or other identificational info. If logical and TRUE then print variable name or character representation of expression. If non-zero length character, then print this value. If FALSE (<i>default</i>) or zero-length character, then no header for printing.
...	Passed to format function
ref	Character or NULL. If character, then checking of ursaRaster sub-class(es):

Pattern	Description
NULL	ursaRaster
(raster brick ursa)	ursaRaster
grid	ursaGrid
(ct color table)	ursaColorTable
stack	ursaStack
con	ursaConnection
val	ursaNumeric OR ursaCategory
cat	ursaCategory

Details

`is.ursa()` is designed mainly for developers to check arguments' class in function's call. `is_ursa` is a synonym to `is.ursa`.

Structure of `ursaRaster` class is generated by non-public `.raster.skeleton()` function.

Value

`ursaRaster` is R's S3 class. It is a list with items:

grid	Geospatial properties. ursaGrid object
con	Connection properties. ursaConnection object
value	2-dimensional numerical or integer matrix of classes ursaValue in (<i>spatial</i> , <i>temporal</i>) specification formed from (<i>samples*lines</i> , <i>bands</i>). If data are not in memory, then NA.
dim	Dimension of <code>\$value</code> . If bands are interpreted as observations in time, then it is <i>spatial</i> by <i>temporal</i> dimension of data. Even data are not in memory, <code>dim</code> is a dimension of whole data.
name	Band names
colortable	Color table. ursaColorTable object

`is.ursa(x)` returns TRUE, if class of `x` is `ursaRaster`

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```

session_grid(NULL)
print(methods(class="ursaRaster"))

a <- pixelsize()
print(a)
print(a,grid=TRUE)
s <- substr(as.character(sessionInfo()),1,48)
b <- rep(a,length(s))
bandname(b) <- s
print(b)

require(datasets)
data(volcano)
print(is.ursa(a))
print(is.ursa(volcano))
print(is.ursa(as.ursa(volcano)))

```

ursaStack	<i>List of raster images.</i>
-----------	-------------------------------

Description

Functions to create list (layers) of multiband raster images (*stack* in the notation of **raster** package) and to coerce list of images to single multiband image (*brick* in the notation of **raster** package).

Usage

```

ursa_stack(...)

ursa_brick(obj)

ursa_apply(obj, FUN, ...)

## S3 method for class 'ursaRaster'
as.list(x, ...)

## S3 method for class 'ursaStack'
unlist(x, recursive, use.names)

```

Arguments

obj, x	Object of class ursaRaster or list of ursaRaster objects. In function ursa_apply argument "X", which is passed to function lapply .
FUN	Argument "FUN", which is passed to function lapply .
recursive	Not used. For consistency with generic function unlist .
use.names	Not used. For consistency with generic function unlist .

... Denending of functions:

ursa_stack, as.list	List of ursaRaster objects
ursa_apply	Arguments "...", which are passed to function lapply .

Details

`as.list` (of ursaRaster object(s)), `ursa_stack` create list of ursaRaster objects, where items of list are sinle-band images. If `x` is ursaRaster object, then `list(x)` create a list of length one, which item is multiband image.

`unlist` (for list of ursaRaster objects), `ursa_brick` create single multiband ursaRaster object. There is an alternative way for unlisting the list of ursaRaster: [as.ursa](#).

Raster *stack* is a way to group bands, for example, by units (degree Celsius, meters).

Raster *brick* is a way to combine versalite images to the single multiband image, for example, for saving in file.

Value

`ursa_stack`, `as.list` return object of class ursaStack. It is a list, with class "ursaStack" attribute.

`unlist` (for list of ursaRaster objects), `ursa_brick` return object of class ursaRaster.

`ursa_apply` returns object of class ursaStack, if result is list of ursaRaster objects, otherwise returns general [list](#).

Warning

There is no any verifications, that [grids](#) of ursaRaster objects are the same.

Note

Generic [unlist\(x\)](#) deals only with class of `x`, but doesn't take into account class of objects in list (e. g., `x[[1]]`). So, there is no effective way to use only [list/unlist](#) for ursaRaster objects to do a conversion between raster *brick* and *stack*. Generic [unlist\(x\)](#) deals only with class of `x`, but doesn't take into account class of objects in list (e. g., `x[[1]]`). So, there is no effective way to use only [list/unlist](#) for ursaRaster objects to do a conversion between raster *brick* and *stack*.

Author(s)

Nikita Platonov

References

<https://CRAN.R-project.org/package=raster>

See Also[lapply](#) [list](#) [unlist](#)[c](#) for ursaRaster objects.**Examples**

```

session_grid(NULL)
a <- ursa_dummy(3)
print(a)
b1 <- ursa_stack(a[1:2],colorize(a[3],ramp=FALSE))
print(b1)
b2 <- as.list(a)
print(b2)
b3 <- list(a[1],a[2:3])
print(b3)
b31 <- lapply(b3,colorize,ramp=FALSE)
print(b31)
b32 <- ursa_apply(b3,colorize,ramp=FALSE,rev=TRUE)
print(b32)
s311 <- ursa_apply(b31,ursa_colortable)
print(s311)
s21 <- lapply(b2,global_mean)
print(s21)
s22 <- sapply(b2,global_mean)
print(s22)
s31 <- lapply(b3,global_mean)
print(s31)
s32 <- sapply(b3,global_mean)
print(s32)
c1 <- unlist(b1)
print(c1)
c2 <- unlist(b2)
print(c2)
c3 <- unlist(b3)
print(if (is.ursa(c3)) c3 else "broken object")
d3 <- as.ursa(b3)
print(if (is.ursa(d3)) d3 else "broken object")

```

ursaValue

Values of raster images.

Description

Class ursaValue is a part of class ursaRaster. It contains values of image. In the case of numeric values, the exterior class is ursaNumeric. In the case of categorical values, the exterior class is ursaCategory.

Usage

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

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

ursa_value(obj, band)
ursa_value(obj, band) <- value
```

Arguments

x	Object of the one of classes <code>ursaNumeric</code> or <code>ursaCategory</code>
...	Further arguments passed to generic functions <code>print</code> and <code>str</code> .
obj	Object of class <code>ursaRaster</code>
band	Optional. Vector of band numbers (positive integer) or band names (character).
value	Numeric or integer scalar, vector, or matrix. Coerced to dimension of <code>ursaValue</code> .

Details

Try to use high-level assignment using replacement `[<-` operator for class `ursaRaster`. However, if you don't get desired result, you can downgrade the level of your code.

Value

Object of class `ursaNumeric` is a numerical matrix. Object of class `ursaCategory` is an integer matrix. Dimensions of this matrix:

<code>dim(...)[1]</code>	Spatial domain; the length is multiplications of lines (rows) and samples (columns)
<code>dim(...)[2]</code>	Band or temporal domain; the length is number of bands

It is allowed to use scalar value `NA` in the case when values are not in memory. In this case the class is `ursaValue`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

Extract `[` and replacement `[<-` methods for class `ursaRaster`

Examples

```

session_grid(NULL)
session_grid(regrid(mul=1/4))
a1 <- create_envi("exam1.envi",bandname=c("today","tomorrow"))
str(ursa_value(a1))
close(a1)
envi_remove("exam1")
a2 <- ursa_dummy(nband=4,min=1,max=99)
str(ursa_value(a2),digits=3)
a3 <- as.integer(a2)
str(ursa_value(a3))
str(ursa_value(a3,2))
print(ursa_value(a3))
print(a3)
ursa_value(a3,"Band 2") <- 199
ursa_value(a3)[,3] <- 299
a3[4] <- 399
print(a3)
ursa_value(a3[1:3]) <- ursa_value(a3[4])
print(a3)
ursa_value(a3[1:3]) <- -c(1:3)
print(a3)

```

ursa_cache

Cache management of ursa package

Description

This help topic is about how cache is managed in the package.

Usage

```
ursa_cache()
```

Details

Users, who want to keep cache files between R sessions, should define [option](#) with name `ursaCacheDir` and value of the path for storage of cache files. This setting can be specified in `~/.Rprofile` file, or be in your code. If you specify permanent cache directory as sub-directory of `tempdir()` (see example), it will be removed after finishing of R session.

Value

NULL

Note

There is no necessary to call this function. It just defines this help topic.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
## internet connection is required
options(ursaCacheDir=file.path(tempdir(),".ursaCacheDir"))
print(c(tempdir=tempdir(),ursaCacheDir=getOption("ursaCacheDir")))
glance("Mount Eden",place="park")
dir(getOption("ursaCacheDir"))
```

ursa_crop	<i>Crop 'no data' margins.</i>
-----------	--------------------------------

Description

Function ursa_crop makes such spatial subset of source raster image, where margins of 'no data' values are absent or have specified width.

Usage

```
ursa_crop(obj, condition, border = 0, expand = 1, resetGrid = TRUE, verbose = FALSE)
```

Arguments

obj	Object of class ursaRaster
condition	Object of class ursaRaster or missing . The condition for cutting. If 'missing' then condition is defined from obj.
border	Integer of length 1, 2 or 4. Desired margins for geographical subset. Units are cells (pixels).
expand	Numeric of length 1, 2 or 4. Desired boundary expansion for geographical subset. Units is ratio (relative to 1). Default is 1.
resetGrid	Logical. If resetGrid=TRUE then sessional grid parameters is established from grid parameters of created raster. If resetGrid=FALSE then sessional grid parameters keep without change. Default is TRUE.
verbose	Logical. TRUE provides some additional information on console.

Details

This function calls [regrid](#) with passing values of arguments resetGrid and verbose without changes.

Bordering (argument border) is applied before expansion (argument expand).

This function is an instrument for data compression for spatial matrices with wide margins of 'no data' value. It keeps spatial structure (pixel's neighborhood) in the internal data storage. Otherwise, [compress](#) reduces object size using spatial indexing with dropping of spatial structure.

Value

Object of class ursaRaster

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
'printCR' <- function(obj) print(with(ursa_grid(obj),c(c=columns,r=rows)))
g0 <- session_grid()
a <- pixelsize()
th <- with(ursa_grid(a),resx*resy*1e-6)
a0 <- a[a>th*0.9]
print(session_grid())
printCR(a0)
print(a0)
a1 <- ursa_crop(a0,resetGrid=TRUE)
print(session_grid())
printCR(a1)
print(a1)
a2 <- ursa_crop(a0,resetGrid=FALSE)
print(session_grid())
printCR(a2)
print(a2)
a3 <- a[a>=th*0.85 & a<=th*1.01]
b1 <- ursa_dummy(nband=3,min=0,max=255)
print(b1)
b2 <- ursa_crop(b1[a3>0],border=10)
print(b2)
printCR(b2)
b2[is.na(b2)] <- 255
display_rgb(b2)
b3 <- ursa_crop(b1,a3,border=0)
print(b3)
printCR(b3)
```

ursa_crs

Extract and assign projection of raster images.

Description

Functions manipulate with \$crs item of the [ursaGrid](#) object, which is embedded in the [ursaRaster](#) object (obj\$grid\$crs). Projection is specified in PROJ.4 notation.

Usage

```
ursa_crs(obj)
ursa_crs(obj, keepGrid = FALSE) <- value
```

Arguments

obj	ursaRaster object. It is allowed ursaGrid object for ursa_proj (<i>Extract</i>) function.
keepGrid	Logical. Should sessional grid be changed after assignment. Default is FALSE.
value	Character sting in PROJ.4 format.

Details

Boath *Extract* and *Replace* functions ursa_proj() and ursa_proj4() are synonyms for ursa_crs.

Value

Extract function ursa_crs returns character value of \$grid\$crs item of ursaRaster object.

Replace function ursa_crs<- returns ursaRaster with modified \$grid\$crs item.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- ursa_dummy(nband=1)
print(ursa_crs(a))
p4s <- "+init=epsg:3576"
ursa_crs(a) <- p4s
print(ursa_crs(a))
fname <- tempfile()
write_envi(a,fname)
a2 <- read_envi(fname,resetGrid=TRUE)
print(ursa_crs(a2))
# try(print(rgdal::CRSargs(sp::CRS(p4s)))) ## 'rgdal' is retired
envi_remove(fname)
```

 ursa_dummy

Generate raster image for examples.

Description

ursa_dummy returns georeferenced raster image with required number of bands. The value of such image has no sence in reality, but are suitable for R's examples.

Usage

```
ursa_dummy(nband = 3L, minvalue = 0, maxvalue = 255, mul = 1, elements = 8L,
           bandname = NULL, nodata = TRUE, resetGrid=FALSE)
```


Arguments

nband	Positive integer. Number of bands. Default is 3L.
minvalue	Numeric of length 1. Minimal value for raster image. Default is 0.
maxvalue	Numeric of length 1. Maximal value for raster image. Default is 255.
mul	Positive numeric. The scaling of the existing session grid . Value 1 means the actual pixel size. Value <1 decreases image size by increasing cell size. Value >1 decreases image size by increasing cell size. Default is 1.
elements	Positive integer. Maximal dimension of matrix, which is proportional to session grid . If elements has small value then the resulting image is smooth, like low-resolution image. The elements has big value, then the resulting image is like white noise.
bandname	Character vector or NULL. Band names for created raster image. If NULL, then band names are generated automatically. Default is NULL.
nodata	Numerical or logical. Set value, which is interpreted as 'no-data' flag. If logical and FALSE then no no-data flag is assigned. If logical and TRUE then value of no-data flag is generated automatically. If numeric, then no-data is assigned to value of this argument. Default is TRUE.
resetGrid	Logical. Whether the grid will be reset to default before raster generation? If FALSE then raster is generated in the sessional grid. If TRUE then default parameters are used for raster and session. Default is FALSE.

Details

Currently, the values are generated using [runif](#).

The value `mul<1` speeds up raster generation.

Value

Object of class `ursaRaster`

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a1 <- as.integer(ursa_dummy(nband=1,mul=1/16,elements=1e3)) ## white noise
## to avoid over-time during example check -- begin
  display(a1,legend=NULL)
## to avoid over-time during example check -- end
a2 <- ursa_dummy()
print(a2)
display_brick(a2,decor=FALSE)
display_stack(a2,decor=FALSE)
display_rgb(a2,decor=FALSE)
```

ursa_grid

Extract and assign spatial parameters of raster images.

Description

Raster image (ursaRaster) contains embedded spatial parameters ([ursaGrid](#)) in item \$grid. These functions manipulate with item \$grid.

Usage

```
ursa_grid(obj)
ursa_grid(obj) <- value
```

```
ursa_ncol(obj)
ursa_nrow(obj)
ursa_columns(obj)
ursa_rows(obj)
ursa_samples(obj)
ursa_lines(obj)
```

```
ursa_extent(obj)
ursa_bbox(obj)
```

```
consistent_grid(obj, ref, expand = 1, border = rep(0, 4), verbose = FALSE)
```

Arguments

obj	ursaRaster object. For ursa_grid function list of ursaRaster objects is allowed.
value	ursaGrid object.
ref	ursaGrid reference object.
expand	Positive numeric. Multiplier of boundary box.
border	integer of length 1 or 4. If length 4, then vector (bottom, left, top, right) in cells for extent expand. If length <4, then value is repeated for length 4. Passed to regrid() .
verbose	Logical. Some output in console. Primarily for debug purposes.

Details

ursa_grid<- may used to minor corrections of spatial parameters. However, it seems that this function is not claimed in practice.

ursa_ncol, ursa_columns, ursa_samples are synonyms for extracting number of columns/samples.

ursa_nrow, ursa_rows, ursa_lines are synonyms for extracting number of rows/lines.

ursa_extent, ursa_bbox, are synonyms for extracting boundary box (spatial extent).

consistent_grid transforms dimension (ursa_nrow() by ursa_ncol()) obj-grid to dimension of ref-grid. This helpful for multipanel plotting if objects have different boundary boxes.

Value

ursa_grid return value of \$grid item of ursaRaster object.
 ursa_grid<- return ursaRaster with modified \$grid item.
 ursa_ncol, ursa_columns, ursa_samples return integer of length 1.
 ursa_nrow, ursa_rows, ursa_lines return integer of length 1.
 ursa_extent, ursa_bbox return numeric of length 4 (xmin, ymin, xmax, ymax).
 ursa_consistent returns [ursaGrid](#) object.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
a <- pixelsize()
print(ursa_grid(a))
ursa_grid(a)$crs <- gsub("\\.0+", "", ursa_grid(a)$crs)
print(ursa_grid(a))
```

ursa_info	<i>Print metadata for raster image.</i>
-----------	---

Description

Function shows information about raster CRS, data type, storage mode, nodata value, structure of band names.

Usage

```
ursa_info(obj, detail = NA, ...)
```

Arguments

obj	ursaRaster object
detail	<i>Not used. Reserved for potential detail levels</i>
...	Arguments, which are passed to str .

Details

ursa_info generates a list and then shows structure of this list via function [str](#).

Value

Object of *temporal* class `ursaMetadata` is a list with items:

<code>columns</code>	Number of columns (samples)
<code>rows</code>	Number of rows (lines)
<code>resx</code>	Grid cell size by horizontal axis
<code>resy</code>	Grid cell size by vertical axis
<code>minx</code>	Left margin of boundary box
<code>maxx</code>	Right margin of boundary box
<code>miny</code>	Bottom margin of boundary box
<code>maxy</code>	Top margin of boundary box
<code>proj4</code>	PROJ.4 string
<code>nodata</code>	<i>Optional.</i> Value, which is interpreted as NA
<code>datatype</code>	<i>Optional.</i> If data are on disk, then integer code of data type.
<code>interleave</code>	<i>Optional.</i> If data are on disk, then abbreviation of bands interleave.
<code>mode</code>	Character of length 2: storage mode and class of value. If data has not been read, then class is "logical". If data is not in memory, then storage mode is "raw".
<code>bandname</code>	Band names.
<code>colortable</code>	<i>Optional.</i> Structure of color table .

Function returns NULL.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[str](#)
[print.ursaRaster](#)

Examples

```
session_grid(NULL)
a <- as.integer(round(ursa_dummy(nband=3)))

print(a) ## print data
ursa_info(a,digits=1) ## print metadata

fname <- tempfile()
write_envi(a,fname,compress=FALSE)
b1 <- open_envi(fname)
ursa_info(b1)
close(b1)
b2 <- read_envi(fname)
```

```

ursa_info(b2)

# print ENVI header
sapply(c(" ----- begin -----",readLines(paste0(fname,".hdr"))
, " ----- end -----"),message)

envi_remove(fname)

```

ursa_match

Raster value matching

Description

ursa_match creates raster mask, which is corresponded to vector of reference values

Usage

```

## S3 method for class 'ursaRaster'
mtfrm(x)

ursa_match(obj, value)

```

Arguments

x, obj	Object of class ursaRaster
value	Vector of reference values, generally, numeric. If obj is categorical, then values can be character for matching to category names.

Details

ursa_match(obj,c("blue","yellow")) is equal to obj=="blue" | obj=="yellow"

Value

Object of class ursaRaster as a mask of correspondence to reference values

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

Comparison `==` in [groupGeneric](#)

Examples

```
session_grid(NULL)
a <- colorize(ursa_dummy(1),name=c("low","medium","high"),stretch="equal")
ta <- as.table(a)
ta
c('category value for "low"'=category_value(a,"low"))
res <- c('match to single char'=ursa_match(a,c("medium"))
        , 'match to multiple num'=ursa_match(a,c(0,category_value(a,"high"))))
        , 'match to multiple char'=ursa_match(a,c("low","high"))
        , 'logical OR' = a=="low" | a=="high")
res
c(band_n(res)[1]==ta[2],band_n(res)[3]==sum(ta[c(1,3)]))
```

ursa_new

Create raster image in memory

Description

ursa_new creates object of class ursaRaster in memory using [session grid parameters](#) or properties of input object ([matrix](#) or [array](#)). By option, [band names](#) and [ignore values](#) are specified.

Usage

```
ursa_new(...)
```

Arguments

- ... Set of arguments, which are recognized via their names (using [regular expressions](#)) and classes:
- value Pattern is "(|^value)". Admissible classes are ([matrix](#), [array](#), [numeric](#), [logical](#)). Values to fill image. Array or matrix defines raster grid. If value=FALSE (logical), then created raster image has no values. By default, value=NA, the created raster image is filled by blank values (NA).
- nband Positive integer. Number of bands. Default is 1L.
- bandname Character. Band names. Default is NULL. If specified, then nband is ignored, and the number of bands is equal to length of bandname character vector.
- ignorevalue Integer or numeric. Value in ENVI binary file, which is interpreted as NA in R.
- datatype Positive integer c(1L, 2L, 3L, 4L, 5L, 11L, 12L, 13L) or character. Data type (integer, floating-point) and byte length. See details for argument datatype of function [create_envi](#). Required for writing raster to ENVI binart file. Optional for rasters in memory. Default is NA: data type is defined internally.
- colortable Object of class [ursaColorTable](#). Color table for raster. Default is NULL: color table is absent

`permute` Logical. Should dimensions of input matrix be changed. Default is FALSE.

`flip` Logical. Vertical flip for input matrix. Default is FALSE: no flip.

`crs` Character or object of class `ursaGrid`. The reference grid for raster's cells. Default is NULL: the grid is defined either from matrix/array structure or from [sessional parameters](#).

`verb(ose)*` Logical. Value TRUE may provide some additional information on console. Default is FALSE.

Details

`ursa_new` creates `ursaRaster` object in memory. To manipulate with raster chunks use the followed construction:

```
a <- create_envi(fname,...)
a[condition_1] <- value
print(a[condition_2])
...
close(a)
```

`ursa_new` is designed to create blank raster images. Use [as.ursa](#) for conversion R objects to `ursaRaster`.

Value

Object of class `ursaRaster`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[as.ursa](#), [create_envi](#).

Examples

```
session_grid(NULL)
a1 <- ursa_new(volcano)
print(a1)
## to avoid over-timing during tests -- begin
  display(a1)
## to avoid over-timing during tests -- end

a2 <- ursa_new(volcano,flip=TRUE)
print(a2)
## to avoid over-timing during tests -- begin
  display(a2)
## to avoid over-timing during tests -- end
```

```

a3 <- ursa_new(volcano,permute=TRUE)
print(a3)
## to avoid over-timing during tests -- begin
  display(a3)
## to avoid over-timing during tests -- end

dima <- c(200,300,4)
b1 <- as.ursa(array(runif(prod(dima)),dim=dima))
print(b1)
display_brick(b1,scale=1,pal=c("white","black"),decor=FALSE)

session_grid(NULL)

c1 <- ursa_new(seq(3))
print(c1)
c2 <- ursa_new(seq(3),bands=3)
print(c2)

c3 <- ursa_new(value=FALSE)
str(ursa_value(c3))

c4 <- ursa_new(bands=2,nodata=-99L)
print(c4)
print(ignorevalue(c4))

c5 <- ursa_new(bandname=format(Sys.Date()+seq(7)-1,"%A"))
ursa_value(c5) <- rev(seq(nband(c5)))
c5 <- colorize(c5)
ct <- ursa_colortable(c5)
print(c5)

v <- ursa_value(c5[3:5])
str(v)
v <- c(v)
str(v)
c6 <- ursa_new(v,colortable=ct)
print(c6)
print(ursa_colortable(c6))

```

whiteboxing

Wrapper to WhiteboxTools

Description

Wrapper to tools from "**whitebox**" package to manipulate with ursaRaster objects

Usage

```
whiteboxing(tool_name, ...)
```


Arguments

`tool_name` Either tool name of Whitebox or function name from [whitebox](#).
`...` List of parameters.

Details

Wrapper to function [wbt_run_tool](#).

ursaRaster object foo can be passed via parameter `input=foo` instead of GeoTIFF file name.

Value

If argument `output` is missed or `output=FALSE`, then object of class `ursaRaster`. Otherwise, output GeoTIFF file name.

Note

Internally, for piping support, first character argument without `*.tif` suffix is interpreted as `tool_name`. First unnamed character argument with `*.tif` suffix or `ursaRaster` object is interpreted as `input`.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[WhiteboxTools](#) [whitebox](#)

Examples

```
if ((requireNamespace("whitebox"))&&(isTRUE(whitebox::wbt_init()))){
  dem <- whitebox::sample_dem_data()
  a1 <- c(DEM=read_gdal(dem))
  a2 <- whiteboxing("BreachDepressions",input=a1)
  b <- list(value=c(a1,a2),difference=c(diff=a1-a2))
  print(b)
  display(b,layout=c(2,NA),legend=list(list("first","left"),list("last","left")))
  if (getRversion()>="4.1.0") {
    a5 <- dem |>
      whiteboxing("feature_preserving_smoothing",filter=9) |>
      whiteboxing("breach_depressions") |>
      print()
  }
}
```

write_envi

Write raster image to ENVI .hdr Labelled Raster file.

Description

write_envi writes in-memory object of class `ursaRaster` to disk in the ENVI .hdr Labelled Raster file format.

Usage

```
write_envi(obj, ...)
```

Arguments

<code>obj</code>	Object of class <code>ursaRaster</code> .
<code>...</code>	Arguments, which are passed to <code>create_envi</code> . Usually, only file name (character) is required. If <code>missing</code> , then occasional name is assigned.

Details

write_envi implements writing the whole `ursaRaster` object to disk. For multiple access to disk (by chunks), use followed construction:

```
a <- create_envi(fname)
a[condition_1] <- value1
a[condition_2] <- value2
...
close(a)
```

Value

Integer code of ENVI data type. See values of the “data type” field in description of the **ENVI Header Format**.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

`create_envi`, *Replace* method [`<-` for `ursaRaster` object, `close_envi` (`close` for `ursaRaster` object).

`write_gdal(..., driver="ENVI")` uses GDAL (**rgdal**) for writing `ursaRaster` object to the ENVI .hdr Labelled Raster file.

Examples

```
session_grid(NULL)
dir.create(tmpWD <- file.path(tempdir(),"certain"))
wd <- setwd(tmpWD)
print(c('temp dir'=session_tempdir(),'working dir'=getwd()))
list1a <- envi_list(session_tempdir())
list1b <- envi_list()
fname <- tempfile(tmpdir=".")
a <- ursa_dummy()
bandname(a) <- c("first","second","third")
write_envi(a)
write_envi(a,fname)
list2a <- envi_list(session_tempdir())
list2b <- envi_list()
fname1 <- list2a[!(list2a %in% list1a)]
fname2 <- list2b[!(list2b %in% list1b)]
print(c('in temp dir'=fname1,'in working dir'=fname2))
a2 <- open_envi(fname1)
print(a2)
close(a2)
envi_remove(c(fname1,fname2))
setwd(wd)
```

write_gdal

Write raster image to GDAL file(s)

Description

write_gdal writes in-memory object of class `ursaRaster` to disk using GDAL from **rgdal** package.

Usage

```
write_gdal(obj, ...)
ursa_write(obj, fname, ...)
```

Arguments

<code>obj</code>	Object of class <code>ursaRaster</code> .
<code>...</code>	Arguments, which are passed to <code>create_gdal</code> . Usually, only file name with extension (character) is required. If extension is ".envi", then GDAL driver "ENVI" is used. If extension is ".tif", then GDAL driver "GTiff" is used. If extension is ".img", then GDAL driver "HFA" is used. If extension is ".jpg" or ".jpeg", then GDAL driver "JPEG" is used. If extension is ".bmp", then GDAL driver "BMP" is used. If extension is ".png", then GDAL driver "PNG" is used. Additionally, argument <code>driver</code> should be specified. If argument <code>...</code> is missing , then occasional name is assigned.

For GDAL formats it is creation options "-co", e. g., compress="LZW", tiled="NO" are interpreted as -co "COMPRESS=LZW" -co "TILED=NO".

For GDAL formats options= (named list list(foo="bar1", foo2="bar2"), named characters c(foo="bar1", foo2="bar2"), characters in format "foo1=bar1 foo2=bar2") is interpreted as creation options (-co) explicitly.

For GDAL formats driver= is interpreted as driver short name (-fo) explicitly.

fname Character. File name with extension.

Details

ursa_write is simplified call of write_gdal.

write_gdal implements writing the whole ursaRaster object to disk. For multiple access to disk (by chunks), use followed [Replace](#) construction:

```
a <- create_gdal(fname)
a[condition_1] <- value1
a[condition_2] <- value2
...
close(a)
```

Value

Integer code of ENVI data type. See values of the "data type" field in description of the [ENVI Header Format](#).

Author(s)

Nikita Platonov <platonov@sev-in.ru>

See Also

[create_gdal](#), [Replace](#) method [[<-](#) for ursaRaster object, [close](#) method for ursaRaster object.
[write_envi](#)

Examples

```
session_grid(NULL)
ftemp <- tempfile(pattern="", fileext="")
fpath <- dirname(ftemp)
fname <- basename(ftemp)
a <- round(ursa_dummy(1,min=0,max=255,nodata=NA))
write_envi(a,file.path(fpath,paste0(fname,"_1",".envi")))
write_gdal(a,file.path(fpath,paste0(fname,"_2")))
write_gdal(a,file.path(fpath,paste0(fname,"_3",".tif")))
write_gdal(a,file.path(fpath,paste0(fname,"_4")),driver="EHdr")
flist <- dir(path=fpath,pattern=fname,full.names=TRUE)
file.remove(flist)
blist <- basename(flist)
res <- NULL
```

```
for (i in seq(4))
  res <- c(res,paste(grep(paste0("_",i),blist,value=TRUE),collapse=" "))
print(res)
```

zonal_stat	<i>Zonal statistics for raster maps</i>
------------	---

Description

'Zonal' operator of map algebra. Applied to raster images.

Usage

```
zonal_stat(x, by, FUN, table = FALSE)

## S3 method for class 'ursaRaster'
aggregate(x, by, FUN, table = FALSE, ...)
```

Arguments

x	ursaRaster object. Image for analysis.
by	ursaRaster object. Image of grouping elements.
FUN	a function to compute the summary statistics which can be applied to all data subsets.
table	Logical. If table=TRUE then summary statistics for each group is returned. The statistics is defined by FUN. If stat=FALSE then result is presented as ursaRaster object.
...	Other arguments which passed to function aggregate of package stats

Details

zonal_stat is a wrapper of `aggregate(x,by,FUN,table=FALSE,na.rm=TRUE)`

You can use multichannel image (argument x) for analysis.

You can use multichannel raster image for group elements (argument by)

Value

If table=FALSE then ursaRaster object of summarized statistics.

If table=TRUE then data.frame.

Author(s)

Nikita Platonov <platonov@sev-in.ru>

Examples

```
session_grid(NULL)
session_grid(regrid(mul=1/2))
a <- pixelsize()
val <- c(normal=a, half=a/2)
gr <- c(group=colorize(a, nbreak=1, lazyload=FALSE))##+0
print(as.table(gr))
##~ display(gr)
ra <- round(aggregate(val, gr, mean), 4)
print(ra)
print(as.table(ra[1]))
print(as.table(ra[2]))
da <- aggregate(val, gr, table=TRUE, mean)
n <- aggregate(a, gr, table=TRUE, length)[, 2, drop=FALSE]
da <- cbind(da, n=unname(n))
gr2 <- c(group2=colorize(a, nbreak=6, lazyload=FALSE))##+0
mgr <- list(gr, gr2)
da2 <- aggregate(val[1], mgr, table=TRUE, mean)
print(da2)
da3 <- aggregate(val, mgr, table=TRUE, mean)
print(da3)
ra3 <- aggregate(val, mgr, table=FALSE, mean) ## not implemented for rasters
print(ra3)
```

Index

* **aplot**

- legend_align, 91
- legend_colorbar, 93
- legend_mtext, 96
- panel_annotation, 106
- panel_coastline, 108
- panel_contour, 112
- panel_decor, 115
- panel_graticule, 116
- panel_new, 119
- panel_plot, 122
- panel_raster, 124
- panel_scalebar, 126
- panel_shading, 129

* **attribute**

- as.ursa, 16
- blank, 24
- identify, 86
- ignorevalue, 89
- pixelsize, 130
- spatial_engine, 152
- spatial_levelsplit, 157
- spatial_read, 158
- spatial_write, 159
- trackline, 165
- ursa, 166
- ursa_crs, 183
- ursa_grid, 186
- ursa_info, 187

* **category**

- reclass, 138

* **classes**

- colortable, 33
- ursaConnection, 169
- ursaCRS, 171
- ursaGrid, 172
- ursaRaster, 175
- ursaStack, 177
- ursaValue, 179

* **color**

- colorize, 30
- cubehelix, 54
- discolor, 57

* **connection**

- create_envi, 51
- open_envi, 103
- open_gdal, 104
- read_envi, 135
- read_gdal, 137
- write_envi, 194
- write_gdal, 195

* **datagen**

- ursa_dummy, 184
- ursa_new, 190

* **dplot**

- compose_design, 39

* **environment**

- session, 149
- ursa_cache, 181

* **file**

- envi_files, 64

* **hplot**

- compose_close, 37
- compose_legend, 43
- compose_open, 45
- compose_panel, 49
- compose_plot, 50
- display, 58
- display_brick, 60
- display_rgb, 61
- display_stack, 62
- glance, 76

* **iteration**

- chunk, 26

* **manip**

- whiteboxing, 192

* **methods**

- as.array, 6

- as.data.frame, 7
- as.integer, 9
- as.matrix, 10
- as.Raster, 12
- as.raster, 14
- as.table, 15
- bandname, 20
- c, 25
- close, 28
- dim, 56
- Extract, 65
- groupGeneric, 82
- head, 84
- hist, 85
- is.na, 90
- na.omit, 101
- nband, 102
- plot, 132
- rep, 144
- Replace, 145
- seq, 147
- sort, 151
- summary, 161
- ursa_match, 189
- * **misc**
 - codec, 29
 - ursaProgressBar, 173
- * **package**
 - ursa-package, 4
- * **print**
 - colortable, 33
 - ursaConnection, 169
 - ursaCRS, 171
 - ursaGrid, 172
 - ursaRaster, 175
 - ursaValue, 179
- * **spatial**
 - allocate, 4
 - as_stars, 19
 - band_group, 21
 - band_stat, 22
 - focal_extrem, 67
 - focal_mean, 69
 - focal_median, 70
 - focal_special, 72
 - get_earthdata, 74
 - global operator, 80
 - local_group, 98
 - local_stat, 100
 - polygonize, 133
 - regrid, 140
 - trackline, 165
 - ursa-package, 4
 - ursa_crop, 182
 - zonal_stat, 197
- * **ts**
 - temporal_interpolate, 162
 - temporal_mean, 163
- * **apply**, 5
 - .as.array (as.array), 6
 - .as.data.frame (as.data.frame), 7
 - .average (local_group), 98
 - .compose_close (compose_close), 37
 - .compose_coastline (panel_coastline), 108
 - .compose_graticule (panel_graticule), 116
 - .glance (glance), 76
 - .legend_colorbar (legend_colorbar), 93
 - .legend_mtext (legend_mtext), 96
 - .panel_annotation (panel_annotation), 106
 - .panel_coastline (panel_coastline), 108
 - .panel_graticule (panel_graticule), 116
 - .panel_new (panel_new), 119
 - .panel_raster (panel_raster), 124
 - .panel_scalebar (panel_scalebar), 126
 - .regrid (regrid), 140
 - [, 66, 105, 136, 180
 - [.ursaColorTable (colortable), 33
 - [.ursaRaster (Extract), 65
 - [<- (Replace), 145
 - [.ursaRaster (as.matrix), 10
 - [], 57
- abbreviate, 95
- abline, 122, 123
- about (ursa-package), 4
- aggregate, 5, 197
- aggregate.ursaRaster (zonal_stat), 197
- allocate, 4, 16, 17
- apply, 83, 164
- array, 6, 16, 146, 190
- as.array, 6, 14
- as.array.ursaRaster (as.array), 6
- as.data.frame, 7, 7, 172
- as.data.frame.ursaGrid (ursaGrid), 172

- as.data.frame.ursaRaster
(as.data.frame), 7
- as.Date, 31
- as.integer, 9, 9
- as.list, 133
- as.list.ursaRaster (ursaStack), 177
- as.matrix, 7, 10, 164
- as.POSIXct, 31
- as.Raster, 12
- as.raster, 7, 12, 14, 14
- as.table, 15, 167
- as.ursa, 11, 16, 138, 164, 167, 178, 191
- as_stars, 19
- as_ursa (as.ursa), 16
- Assign (Replace), 145
- axis, 40
- band names, 190
- band.*, 23
- band_blank, 101, 102
- band_blank (blank), 24
- band_group, 21
- band_max (band_group), 21
- band_mean, 99
- band_mean (band_group), 21
- band_min (band_group), 21
- band_n (band_group), 21
- band_nNA (band_group), 21
- band_quantile (band_group), 21
- band_sd (band_group), 21
- band_stat, 22, 22
- band_sum (band_group), 21
- bandname, 20, 103
- bandname<- (bandname), 20
- base::close, 28
- base::table, 15
- basename, 64
- blank, 24
- box, 48, 122, 123
- brick, 16, 17
- c, 25, 57, 145, 179
- capabilities, 75
- categories, 139
- category_value (colortable), 33
- caTools::runmean, 164
- character, 17, 44
- chunk, 26
- chunk_band (chunk), 26
- chunk_expand (chunk), 26
- chunk_line (chunk), 26
- class, 188
- class-ursaCategory (ursaValue), 179
- class-ursaColorTable (colortable), 33
- class-ursaConnection (ursaConnection), 169
- class-ursaCRS (ursaCRS), 171
- class-ursaGrid (ursaGrid), 172
- class-ursaNumeric (ursaValue), 179
- class-ursaRaster (ursaRaster), 175
- class-ursaStack (ursaStack), 177
- class-ursaValue (ursaValue), 179
- close, 28, 28, 53, 104, 105, 174, 194, 196
- close(), 52
- close.ursaProgressBar
(ursaProgressBar), 173
- close_envi, 53, 136, 194
- close_envi (close), 28
- codec, 29
- colnames, 88
- color table, 125, 139, 188
- Color tables, 132
- color tables, 83, 113
- colorize, 30, 34, 35, 79, 86, 113, 125, 139
- colors, 34
- colortable, 15, 33, 34
- commonGeneric, 36
- Complex.ursaRaster (groupGeneric), 82
- compose_close, 37, 42, 47, 48, 59–63
- compose_coastline (panel_coastline), 108
- compose_design, 39, 45, 47, 58, 60, 61, 63, 106, 118, 121, 127
- compose_graticule (panel_graticule), 116
- compose_legend, 43, 50, 51, 59, 60, 63, 96, 97
- compose_open, 38, 42, 45, 55, 58, 60, 61, 63, 86, 121, 125, 127
- compose_panel, 44, 49, 50, 51, 63
- compose_plot, 50, 50, 58, 60, 61, 63, 118, 127
- compress, 182
- compress (codec), 29
- connections, 52, 53, 103, 104, 170
- consistent_grid (ursa_grid), 186
- contour, 10, 113, 114
- contourLines, 113, 114
- coord_cr (identify), 86
- coord_xy (identify), 86
- create_envi, 24, 51, 103, 104, 190, 191, 194

- create_gdal, [195, 196](#)
- create_gdal (create_envi), [51](#)
- cubehelix, [54](#)
- data.frame, [16, 23](#)
- decompress (codec), [29](#)
- diff, [37](#)
- diff.ursaRaster (commonGeneric), [36](#)
- dim, [56, 167](#)
- dim.ursaGrid (ursaGrid), [172](#)
- dim<-.ursaRaster (dim), [56](#)
- dir, [154, 155](#)
- dirname, [64](#)
- discolor, [57](#)
- display, [58, 61–63, 79, 93, 118, 121, 127, 133](#)
- display_brick, [58, 59, 60, 62, 63, 76, 118, 127](#)
- display_hetero (display_stack), [62](#)
- display_homo (display_brick), [60](#)
- display_rgb, [58, 59, 61, 61, 63, 118, 127](#)
- display_stack, [58, 59, 61, 62, 62, 118, 127](#)
- download.file, [75](#)
- duplicated, [37](#)
- duplicated.ursaRaster (commonGeneric), [36](#)
- envi_copy (envi_files), [64](#)
- envi_exists (envi_files), [64](#)
- envi_files, [64](#)
- envi_list (envi_files), [64](#)
- envi_remove (envi_files), [64](#)
- envi_rename (envi_files), [64](#)
- expression, [107](#)
- Extract, [65, 105, 136, 146](#)
- factor, [8](#)
- file.rename, [65](#)
- filled.contour, [10, 132, 133](#)
- focal_extrem, [67, 71, 74](#)
- focal_max (focal_extrem), [67](#)
- focal_mean, [29, 68, 69, 71, 74, 143](#)
- focal_median, [68, 70, 74](#)
- focal_min (focal_extrem), [67](#)
- focal_special, [72](#)
- format, [175, 176](#)
- get_earthdata, [74](#)
- getwd(), [149](#)
- glance, [76](#)
- global operator, [80](#)
- global_max (global operator), [80](#)
- global_mean, [99](#)
- global_mean (global operator), [80](#)
- global_median (global operator), [80](#)
- global_min (global operator), [80](#)
- global_n (global operator), [80](#)
- global_nNA (global operator), [80](#)
- global_quantile (global operator), [80](#)
- global_range (global operator), [80](#)
- global_sd (global operator), [80](#)
- global_sum (global operator), [80](#)
- grep, [66, 146](#)
- grids, [178](#)
- Group, [101](#)
- groupGeneric, [82, 189](#)
- head, [84](#)
- hist, [85, 85, 86](#)
- histogram(hist), [85](#)
- identify, [86](#)
- ignore values, [190](#)
- ignorevalue, [89, 167](#)
- ignorevalue<- (ignorevalue), [89](#)
- image, [10, 59, 94, 125, 132, 133](#)
- image.ursaRaster (plot), [132](#)
- index (ursa-package), [4](#)
- integer, [9, 59](#)
- is.infinite.ursaRaster (is.na), [90](#)
- is.na, [24, 90](#)
- is.na.ursaRaster (is.na), [90](#)
- is.na<-.ursaRaster (is.na), [90](#)
- is.nan.ursaRaster (is.na), [90](#)
- is.ursa (ursaRaster), [175](#)
- is_spatial (spatial_engine), [152](#)
- is_spatial_lines (spatial_engine), [152](#)
- is_spatial_points (spatial_engine), [152](#)
- is_spatial_polygons (spatial_engine), [152](#)
- is_ursa (ursaRaster), [175](#)
- lapply, [177–179](#)
- layout.text, [108](#)
- legend(), [122](#)
- legend_align, [91, 94](#)
- legend_colorbar, [44, 63, 91, 92, 93, 97, 113, 125](#)
- legend_mtext, [44, 96](#)

- length.ursaRaster (nband), 102
- lines, 122, 123
- List, 159
- list, 10, 16, 17, 25, 40, 45, 58–63, 178, 179, 186
- list.files, 64
- Local, 101
- local_all (local_group), 98
- local_any (local_group), 98
- local_group, 98
- local_length (local_group), 98
- local_max (local_group), 98
- local_mean (local_group), 98
- local_median (local_group), 98
- local_min (local_group), 98
- local_quantile (local_group), 98
- local_sd (local_group), 98
- local_stat, 83, 99, 100
- local_sum (local_group), 98
- local_var (local_group), 98
- logical, 190
- match, 66, 146
- match.arg, 72
- Math, 82
- Math.ursaRaster (groupGeneric), 82
- matrix, 6, 16, 146, 162, 190
- mean, 83
- mean.ursaRaster (local_group), 98
- median, 83, 100
- median.ursaRaster (local_group), 98
- missed, 131
- missing, 54, 131, 134, 182, 194, 195
- mtext, 40, 97
- mtfrm.ursaRaster (ursa_match), 189
- na.omit, 101
- names, 103, 167
- names.ursaColorTable (colortable), 33
- names.ursaRaster (bandname), 20
- names<- .ursaColorTable (colortable), 33
- names<- .ursaRaster (bandname), 20
- nband, 20, 102
- nband(obj), 24
- numeric, 16, 81, 85, 190
- open_envi, 103, 105, 136
- open_gdal, 104
- Ops.ursaRaster (groupGeneric), 82
- option, 181
- options, 42, 47
- palettize (colorize), 30
- panel_abline (panel_plot), 122
- panel_annotation, 49, 50, 59, 60, 63, 106
- panel_box (panel_plot), 122
- panel_coastline, 48–50, 59–61, 63, 108, 116
- panel_contour, 112, 123
- panel_decor, 59–61, 63, 115
- panel_graticule, 49, 50, 59–63, 116, 116
- panel_lines (panel_plot), 122
- panel_new, 49, 50, 58, 60, 61, 63, 119
- panel_plot, 122
- panel_points (panel_plot), 122
- panel_polygon (panel_plot), 122
- panel_raster, 49, 50, 58, 60–63, 124, 129
- panel_scalebar, 49, 50, 59–61, 63, 116, 126
- panel_segments (panel_plot), 122
- panel_shading, 129
- panel_text (panel_plot), 122
- par, 110, 117, 118, 120, 129
- par(las=), 94
- pixelsize, 130
- plot, 59, 121–123, 132
- plot.ursaRaster (plot), 132
- plot.window, 120
- png, 45–47, 85, 94
- points, 122, 123
- polygon, 110, 122, 123, 129
- polygonize, 133
- polypath, 110
- pretty, 32
- print, 23, 34, 169, 171, 172, 180
- print.ursaCategory (ursaValue), 179
- print.ursaColorTable (colortable), 33
- print.ursaConnection (ursaConnection), 169
- print.ursaCRS (ursaCRS), 171
- print.ursaGrid (ursaGrid), 172
- print.ursaNumeric (ursaValue), 179
- print.ursaRaster, 188
- print.ursaRaster (ursaRaster), 175
- quantile, 21, 98, 99
- quantile.ursaRaster (local_group), 98
- range, 129

- raster, [6](#), [7](#), [12](#), [14](#), [16](#), [17](#)
- raster grid, [17](#), [131](#)
- RasterBrick, [12](#)
- rasterImage, [107](#), [125](#)
- rasterize, [5](#)
- RasterLayer, [12](#)
- RasterStack, [12](#)
- read_envi, [17](#), [135](#)
- read_gdal, [17](#), [105](#), [136](#), [137](#)
- read_sf, [158](#)
- readBin, [169](#)
- readPNG, [38](#)
- rearranged, [55](#)
- reclass, [138](#)
- rect, [108](#), [120](#), [121](#)
- regrid, [57](#), [66](#), [107](#), [113](#), [135](#), [140](#), [143](#), [149](#),
[151](#), [173](#), [182](#)
- regrid(), [186](#)
- regular expression, [137](#)
- Regular expressions, [75](#)
- regular expressions, [5](#), [7](#), [10](#), [14](#), [77](#), [82](#),
[87](#), [93](#), [96](#), [106](#), [109](#), [112](#), [117](#), [120](#),
[125](#), [126](#), [135](#), [141](#), [148](#), [155](#), [190](#)
- rep, [144](#)
- Replace, [145](#), [194](#), [196](#)
- reversed, [55](#)
- rownames, [88](#)
- runif, [185](#)
- S3 Generic Functions, [83](#)
- seek, [169](#), [170](#)
- seek.ursaConnection(ursaConnection),
[169](#)
- segments, [122](#), [123](#)
- seq, [147](#)
- series(head), [84](#)
- session, [149](#)
- session grid, [78](#), [131](#), [135](#), [185](#)
- session grid parameters, [190](#)
- session_bbox(session), [149](#)
- session_cellsize(session), [149](#)
- session_crs(session), [149](#)
- session_dim(session), [149](#)
- session_grid, [52](#), [53](#), [103](#), [110](#), [173](#)
- session_grid(session), [149](#)
- session_grid(), [5](#)
- session_pngviewer, [79](#)
- session_pngviewer(session), [149](#)
- session_proj(session), [149](#)
- session_proj4(session), [149](#)
- session_tempdir(session), [149](#)
- session_use_experimental_functions
(session), [149](#)
- sessional grid, [17](#)
- sessional parameters, [191](#)
- setUrsaProgressBar(ursaProgressBar),
[173](#)
- sf, [156](#)
- sf::gdal_read, [17](#)
- sf::st_write, [160](#)
- sfc, [109](#)
- sort, [151](#)
- sp, [156](#)
- Spatial, [109](#)
- spatial_area(spatial_engine), [152](#)
- spatial_basename(spatial_engine), [152](#)
- spatial_bbox(spatial_engine), [152](#)
- spatial_bbox<-(spatial_engine), [152](#)
- spatial_bind, [165](#)
- spatial_bind(spatial_engine), [152](#)
- spatial_buffer(spatial_engine), [152](#)
- spatial_centroid(spatial_engine), [152](#)
- spatial_colnames(spatial_engine), [152](#)
- spatial_colnames<-(spatial_engine), [152](#)
- spatial_coordinates(spatial_engine),
[152](#)
- spatial_count(spatial_engine), [152](#)
- spatial_crop(spatial_engine), [152](#)
- spatial_crs(spatial_engine), [152](#)
- spatial_crs<-(spatial_engine), [152](#)
- spatial_data(spatial_engine), [152](#)
- spatial_data<-(spatial_engine), [152](#)
- spatial_difference(spatial_engine), [152](#)
- spatial_dim(spatial_engine), [152](#)
- spatial_dir(spatial_engine), [152](#)
- spatial_engine, [152](#)
- spatial_fields(spatial_engine), [152](#)
- spatial_fields<-(spatial_engine), [152](#)
- spatial_filelist(spatial_engine), [152](#)
- spatial_geometry(spatial_engine), [152](#)
- spatial_geometry<-(spatial_engine), [152](#)
- spatial_geotype(spatial_engine), [152](#)
- spatial_grid(spatial_engine), [152](#)
- spatial_intersection(spatial_engine),
[152](#)
- spatial_length(spatial_engine), [152](#)
- spatial_levelsplit, [157](#)

- `spatial_ncol` (`spatial_engine`), 152
- `spatial_nrow` (`spatial_engine`), 152
- `spatial_pattern` (`spatial_engine`), 152
- `spatial_proj` (`spatial_engine`), 152
- `spatial_proj4` (`spatial_engine`), 152
- `spatial_proj4<-` (`spatial_engine`), 152
- `spatial_proj<-` (`spatial_engine`), 152
- `spatial_read`, 158, 160
- `spatial_shape` (`spatial_engine`), 152
- `spatial_symdifference` (`spatial_engine`), 152
- `spatial_transform` (`spatial_engine`), 152
- `spatial_trim` (`spatial_engine`), 152
- `spatial_union` (`spatial_engine`), 152
- `spatial_valid` (`spatial_engine`), 152
- `spatial_write`, 158, 159
- `SpatialGridDataFrame`, 16
- `SpatialPixelsDataFrame`, 16
- `SpatialPointsDataFrame`, 16
- `st_buffer`, 154
- `st_write()`, 160
- `stack`, 16, 17
- `stop`, 83
- `storage.mode`, 10, 170
- `str`, 169, 171, 172, 180, 187, 188
- `str.ursaCRS` (`ursaCRS`), 171
- `str.ursaGrid` (`ursaGrid`), 172
- `str.ursaRaster` (`ursaRaster`), 175
- `Subset` (`Extract`), 65
- `subset`, 95
- `Summary`, 82
- `summary`, 161, 161
- `summary.matrix`, 161
- `Summary.ursaRaster` (`groupGeneric`), 82
- `table`, 15
- `tail.ursaRaster` (`head`), 84
- `tempdir()`, 149, 181
- `temporal_interpolate`, 162
- `temporal_mean`, 162, 163, 163
- `text`, 107, 108, 122, 123
- `tkProgressBar`, 174, 175
- `trackline`, 165
- `txtProgressBar`, 174, 175
- `unclassed`, 11
- `unlist`, 17, 177–179
- `unlist.ursaStack` (`ursaStack`), 177
- `update_coastline` (`panel_coastline`), 108
- `ursa`, 166, 170
- `ursa-package`, 4
- `ursa<-` (`ursa`), 166
- `ursa_apply` (`ursaStack`), 177
- `ursa_bbox` (`ursa_grid`), 186
- `ursa_blank` (`blank`), 24
- `ursa_brick`, 26
- `ursa_brick` (`ursaStack`), 177
- `ursa_cache`, 181
- `ursa_color` (`colortable`), 33
- `ursa_colorindex` (`colortable`), 33
- `ursa_colortable`, 33, 167
- `ursa_colortable` (`colortable`), 33
- `ursa_colortable<-` (`colortable`), 33
- `ursa_columns` (`ursa_grid`), 186
- `ursa_crop`, 182
- `ursa_crs`, 183
- `ursa_crs<-` (`ursa_crs`), 183
- `ursa_dummy`, 184
- `ursa_exists` (`envi_files`), 64
- `ursa_extent` (`ursa_grid`), 186
- `ursa_grid`, 167, 186
- `ursa_grid<-` (`ursa_grid`), 186
- `ursa_hist` (`hist`), 85
- `ursa_info`, 23, 167, 187
- `ursa_lines` (`ursa_grid`), 186
- `ursa_match`, 189
- `ursa_ncol` (`ursa_grid`), 186
- `ursa_new`, 16, 17, 24, 53, 190
- `ursa_nodata` (`ignorevalue`), 89
- `ursa_nodata<-` (`ignorevalue`), 89
- `ursa_nrow` (`ursa_grid`), 186
- `ursa_open` (`open_gdal`), 104
- `ursa_proj`, 167
- `ursa_proj` (`ursa_crs`), 183
- `ursa_proj4` (`ursa_crs`), 183
- `ursa_proj4<-` (`ursa_crs`), 183
- `ursa_proj<-` (`ursa_crs`), 183
- `ursa_read` (`read_gdal`), 137
- `ursa_rows` (`ursa_grid`), 186
- `ursa_samples` (`ursa_grid`), 186
- `ursa_seqc` (`seq`), 147
- `ursa_seqr` (`seq`), 147
- `ursa_seqx` (`seq`), 147
- `ursa_seqy` (`seq`), 147
- `ursa_stack`, 25
- `ursa_stack` (`ursaStack`), 177
- `ursa_table` (`as.table`), 15

ursa_value, [164](#), [167](#)
ursa_value (ursaValue), [179](#)
ursa_value<- (ursaValue), [179](#)
ursa_write (write_gdal), [195](#)
ursaCategory, [176](#)
ursaColorTable, [11](#), [176](#), [190](#)
ursaConnection, [168](#), [176](#)
ursaCRS, [171](#)
ursaGrid, [141](#), [142](#), [149–151](#), [172](#), [176](#), [183](#),
[186](#), [187](#)
ursaLayout, [48](#)
ursaLayout (compose_design), [39](#)
ursaNumeric, [176](#)
ursaProgressBar, [173](#)
ursaRaster, [81](#), [137](#), [175](#), [176](#), [183](#)
ursaStack, [176](#), [177](#)
ursaValue, [176](#), [179](#)
ursula (ursa-package), [4](#)

value_cr (identify), [86](#)
value_ll (identify), [86](#)
value_xy (identify), [86](#)

wbt_run_tool, [193](#)
whitebox, [193](#)
whiteboxing, [192](#)
write_envi, [194](#), [196](#)
write_gdal, [194](#), [195](#)
write_sf, [160](#)
writeBin, [169](#)
writePNG, [38](#)

zonal_stat, [197](#)