Package 'raster'

February 27, 2012

Type Package

Title Geographic analysis and modeling with raster data

Version 1.9-70

Date 27-February-2012

Depends methods, sp, R (>= 2.11.0)

Suggests rgdal (>= 0.5-33), rgeos (>= 0.2-1), ncdf, igraph, snow,tcltk

Author Robert J. Hijmans & Jacob van Etten

Maintainer Robert J. Hijmans <r.hijmans@gmail.com>

Description Reading, writing, manipulating, analyzing and modeling of gridded spatial data. The package implements basic and high-level functions, as well as map algebra. Processing of very large files is supported.

License GPL (>= 3)

URL http://raster.r-forge.r-project.org/

Repository CRAN

Date/Publication 2012-02-27 09:28:05

R topics documented:

`raster-package` *An overview of the functions in this package*

Description

The raster package provides classes and functions to manipulate geographic (spatial) data in 'raster' format. Raster data divides space into cells (rectangles; pixels) of equal size (in units of the coordinate reference system). Such data are also referred to as 'grid' data.

The package should be particularly useful when using very large datasets that can not be loaded into the computer's memory. Functions will work correctly, because they they process large files in chunks, i.e., they read, compute, and write blocks of data, without loading all values into memory at once.

Below is a list of functions grouped by theme. See the vignette for more information and some examples (you can open it by running this command: `vignette('Raster')`)

Details

The package implements classes for Raster data (see Raster-class) and supports

- Creation of Raster* objects from scratch or from file
- Handling extremely large raster files
- Raster algebra and overlay functions
- Distance, neighborhood (focal) and patch functions
- Polygon, line and point to raster conversion
- Model predictions
- Summarizing raster values
- Easy access to raster cell-values
- Plotting (making maps)
- Manipulation of raster extent, resolution and origin
- Computation of row, col and cell numbers to coordinates and vice versa
- Reading and writing various raster file types

I. Creating Raster* objects

Raster* objects can be created, from scracth, files, or from objects of other classes, with the following functions:

raster	To create a RasterLayer
stack	To create a RasterStack (multiple layers)
brick	To create a RasterBrick (multiple layers)
addLayer	Add a layer to a Raster* object
dropLayer	Remove a layer from a RasterStack or RasterBrick
unstack	Create a list of RasterLayer objects from a RasterStack

RasterLayer, RasterStack, and RasterBrick objects are, as a group, referred to as Raster* objects

II. Changing the spatial extent and/or resolution of Raster* objects

merge	Combine Raster* objects with different extents (but same origin and resolution)
mosaic	Combine RasterLayers with different extents and a function to set values in overlap areas
crop	Select a geographic subset of a Raster* object
expand	Enlarge a Raster* object
trim	Trim a Raster* object by removing exterior rows and/or columns that only have NA values
aggregate	Combine cells of a Raster* object to create larger cells
disaggregate	Subdivide cells
resample	Warp values to a Raster* object with a different origin or resolution
projectRaster	Warp values to a Raster* object with a different coordinate reference system
shift	Move the location of Raster
flip	Flip values horizontally or vertically
rotate	Rotate values around the date-line (for lon/lat data)
t	Transpose a Raster* object

III. Raster algebra

Arith-methods	Arith functions (+, -, *, ^, %%, %/%, /)
Math-methods	Math functions (abs, sqrt, trunc, log, log10, exp, sin, round among others)
Logic-methods	Logic functions (!, &, |)
Summary-methods	Summary functions (mean, max, min, range, prod, sum, any, all)
Compare-methods	Compare functions (==, !=, >, <, <=, >=)

IV. Cell based computation

calc	Computations on a single Raster* object
overlay	Computations on multiple RasterLayer objects
cover	First layer covers second layer except where the first layer is NA
mask	Use values from first Raster except where cells of the mask Raster are NA (these become NA)

cut	Reclassify values using ranges
subs	Reclassify values using a 'is-becomes' matrix
reclass	Reclassify using a 'from-to-becomes' matrix
init	Initialize cells with new values
stackApply	Computations on groups of layers in Raster* object
stackSelect	Select cell values from different layers using an index RasterLayer

V. Spatial contextual computation

distance	Shortest distance to a cell that is not NA
gridDistance	Distance when traversing grid cells that are not NA
distanceFromPoints	Shortest distance to any point in a set of points
direction	Direction (azimuth) to or from cells that are not NA
focal	Focal (neighborhood; moving window) functions
edge	Edge detection
clump	Find clumps (patches)
adjacency	Identify cells that are adjacent to a set of cells on a raster
area	Compute area of cells (for longitude/latitude data)
terrain	Compute slope, aspect and other characteristics from elevation data
Moran	Compute global or local Moran or Geary indices of spatial autocorrelation

VI. Model predictions

predict	Predict a non-spatial model to a RasterLayer
interpolate	Predict a spatial model to a RasterLayer

VII. Data type conversion

rasterize	Rasterizing points, lines or polygons
rasterToPoints	Create points from a RasterLayer
rasterToPolygons	Create polygons from a RasterLayer
rasterToContour	Contour lines from a RasterLayer
rasterFromXYZ	RasterLayer from regularly spaces points
rasterFromCells	RasterLayer from a Raster object and cell numbers
raster	RasterLayer from SpatialGrid*, image, or matrix objects

You can coerce Raster* objects to Spatial* objects using `as`, as in `as(object, 'SpatialGridDataFrame')`

VIII. Summarizing

`cellStats`	Summarize a Raster cell values with a function
`summary`	Summary of the values of a Raster* object (quartiles and mean)
`freq`	Frequency table of Raster cell values
`count`	Count the frequency of a single (range of) value(s)
`crosstab`	Cross-tabulate two RasterLayer objects
`unique`	Get the unique values in a Raster* object
`zonal`	Summarize a Raster* object by zones in a RasterLayer

IX. Accessing values of Raster* object cells

`getValues`	Get all cell values (fails with very large rasters), or a row of values (safer)
`getValuesBlock`	Get values for a block (a rectangular area defined by start and end row and column
`as.matrix`	Get cell values as a matrix
`as.array`	Get cell values as an array
`extract`	Extract cell values from a Raster* object (e.g., by cell, coordinates, polygon)
`sampleRandom`	Random sample
`sampleRegular`	Regular sample
`minValue`	Get the minimum value of the cells of a Raster* object (not always known)
`maxValue`	Get the maximum value of the cells of a Raster* object (not always known)
`setMinMax`	Compute the minimum and maximum value of a Raster* object if these are not known

You can also use indexing with `[` for cell numbers, and `[[` for row / column number combinations

X. Plotting

Maps

`plot`	Plot a Raster* object. The main method to create a map
`plotRGB`	Combine three layers (red, green, blue channels) into a single 'real color' image
`spplot`	Plot a Raster* with the spplot function (sp package)
`image`	Plot a Raster* with the image function
`persp`	Perspective plot of a RasterLayer
`contour`	Contour plot of a RasterLayer
`filledContour`	Filled contour plot of a RasterLayer
`text`	Plot the values of a RasterLayer on top of a map

The rasterVis packages has additional plotting methods for Raster objects using methods from the

lattice package and from other packages.

Interacting with a map (plot of a RasterLayer)

zoom	Zoom in on a plot of a RasterLayer
click	Draw points and/or query values of RasterLayer by clicking on a map
drawPoly	Create a SpatialPolygons object by drawing it
drawLine	Create a SpatialLines object by drawing it
drawExtent	Create an Extent object by drawing it

Other plots

plot	x-y scatter plot of values of two RasterLayer objects
hist	Histogram of Raster* object values
density	Density plot of Raster* object values
pairs	Pairs plot for layers in a RasterStack or RasterBrick
boxplot	Box plot of the values of one or multiple layers

XI. Getting and setting Raster* dimensions

Basic parameters of existing Raster* objects can be obtained, and in most cases changed, with:

ncol	The number of columns
nrow	The number of rows
ncell	The number of cells (can not be set directly, only via ncol or nrow)
res	The resolution (x and y)
xres	The x resolution (can be set with res)
yres	The y resolution (can be set with res)
xmin	The minimum x coordinate (or longitude)
xmax	The maximum x coordinate (or longitude)
ymin	The minimum y coordinate (or latitute)
ymax	The maximum y coordinate (or latitute)
extent	The extent (minimum and maximum x and y coordinates)
origin	The origin of a Raster* object
projection	The coordinate reference system (map projection)
isLonLat	Test if an object has a longitude/latitude coordinate reference system
filename	Filename to which a RasterLayer or RasterBrick is linked
band	Band (layer) of a multi-band file that this RasterLayer is linked to
nbands	How many bands (layers) does the file have?
compare	Compare for equality the basic parameters of two Raster* objects
NAvalue	Get or set the NA value (for reading from file; see writeRaster for setting it when writing a file)

If there are values associated with a RasterLayer object (either in memory or via a link to a file) these are lost when you change the number of columns or rows or the resolution. This is not the case when the extent is changed (as the number of columns and rows will not be affected). Simlarly, with **projection** you can set the projection, but this does not transform the data (see projectRaster

for that).

XII. Computing row, column, cell numbers and coordinates

Cell numbers start at 1 in the upper-left corner. They increase within rows, from left to right, and then row by row from top to bottom. Likewise, row numbers start at 1 at the top of the raster, and column numbrers start at 1 at the left side of the raster.

xFromCol	x-coordinates from column numbers
yFromRow	y-coordinates from row numbers
xFromCell	x-coordinates from row numbers
yFromCell	y-coordinates from cell numbers
xyFromCell	x and y coordinates from cell numbers
colFromX	Column numbers from x-coordinates (or longitute)
rowFromY	Row numbers from y-coordinates (or latitute)
cellFromXY	Cell numbers from x and y coordinates
cellFromRowCol	Cell numbers from row and column numbers
cellsFromExtent	Cell numbers from extent object
coordinates	x and y coordinates for all cells
validCell	Is this a valid cell number?
validCol	Is this a valid column number?
validRow	Is this a valid row number?

XIII. Writing files

Basic writing

setValues	Put new values in a Raster* object
writeRaster	Write all values of Raster* object to disk
KML	Save raster as KML file

Advanced writing of entire files or parts thereof

blockSize	Get suggested block size for reading and writing
writeStart	Open a file for writing
writeValues	Write some values
writeStop	Close the file after writing
update	Change the values of an existing file

XIV. Extent objects

extent	Create an extent object
intersect	Intersect two extent objects
union	Combine two extent objects
round	round/floor/ceiling of the coordinates of an Extent object

alignExtent	Align an extent with a Raster* object
drawExtent	Create an Extent object by drawing it on top of a map (see plot)

You can also use S4 type coercion such as `as(extent, 'SpatialPolygons')`

XV. Options

setOptions	Set session options
showOptions	Show session options
saveOptions	Save session options to make them persistent
clearOptions	Set session options to default values

XVI. Miscellaneous

getData	Download and geographic data
pointDistance	Distance between points
readIniFile	Read a (windows) 'ini' file
hdr	Write header file for a number of raster formats
trim	Remove leading and trainling blanks from a character string
extension	Get or set the extentsion of a filename
cv	Coefficient of variation
modal	Modal value
sampleInt	Random sample of (possibly very large) range of integer values
showTmpFiles	Show temporary files
removeTmpFiles	Remove temporary files

XVII. For programmers

canProcessInMemory	Test whether a file can be created in memory
pbCreate	Initialize a progress bar
pbStep	Take a progress bar step
pbClose	Close a progress bar
openConnection	Open a file connection
closeConnection	Close a file connection
rasterTmpFile	Get a name for a temporary file
inMemory	Are the cell values in memory?
fromDisk	Are the cell values read from a file?

Acknowledgements

Extensive suggestions were made by Jonathan Greenberg, Agustin Lobo, Matteo Mattiuzzi, Steven Mosher, and Kevin Ummel. Contributions were also made by Neil Best, Andrew Bevan, Roger Bivand, Isabelle Boulangeat, Lyndon Estes, Josh Gray, Tim Haering, Herry Herry, Paul Hiemstra, Ned Hornig, Mayeul Kauffmann, Rainer Krug, Alice Laborte, John Lewis, Lennon Li, Justin McGrath, Richard Plant, Edzer Pebesma, Oscar Perpinan Lamigueiro, Etienne Racine, David Ramsey, Michael Sumner, Stefan Schlaffer, Jon Olav Skoien, Shaun Walbridge, Julian Zeidler, and others.

Author(s)

Robert J. Hijmans and Jacob van Etten

Maintainer: Robert J. Hijmans <r.hijmans@gmail.com>

---

addLayer *Add or drop a layer*

Description

Add a layer to a Raster* object or drop a layer from a RasterStack or RasterBrick. The object returned is always a RasterStack (unless nothing to add or drop was provided, in which case the original object is returned).

Usage

```
addLayer(x, ...)
dropLayer(x, i, ...)
```

Arguments

x	Raster object
i	Indices of the layers to be dropped
...	Additional arguments (none)

Value

RasterStack

Author(s)

Robert J. Hijmans

See Also

subset

Examples

```
file <- system.file("external/test.grd", package="raster")
s <- stack(file, file, file)
r <- raster(file)
s <- addLayer(s, r/2, r*2)
s
s <- dropLayer(s, c(3, 5))
nlayers(s)
```

`adjacency` *Pairs of adjacent cells*

Description

Identify pairs of cells that are adjacent.

Usage

```
adjacency(x, fromCells, toCells, directions)
```

Arguments

`x`	Raster* object
`fromCells`	a vector of cell numbers for which adjacent cells should be calculated
`toCells`	a vector of cell numbers from which adjacent cells are selected. You can use the `adjacent` function if you want all cells to be considered
`directions`	in how many direction cells should be connected: 4, 8 or 16; or "bishop"

Details

Cell numbers start with 1 in the upper-left corner and increase from left to right and from top to bottom.

Number of directions: 4 connects cells with one-cell rook moves, 8 with one-cell queen moves, and 16 with knight and one-cell queen moves. "bishop" connects cells with one-cell diagonal moves.

The function connects the outer meridians for global rasters with a longitude/latitude coordinate reference system.

Value

A two column matrix with each row containing a pair of adjacent cells.

Author(s)

Jacob van Etten <jacobvanetten@yahoo.com>

See Also

adjacent

Examples

```
r <- raster(nrows=10, ncols=10)
adj <- adjacency(r, fromCells = c(1,30,55,72,100),
toCells = 1:ncell(r), directions=4)

# put the result in a RasterLayer
r[] <- 0
r[c(1,30,55,72,100)] <- 1
r[adj[,2]] <- 2
plot(r)


# How many time does one cell value occur next to another cell value?
r <- raster(ncol=10, nrow=10)
set.seed(0)
r[] <- round(runif(ncell(r)) * 5)
a <- adjacency(r, 1:ncell(r), 1:ncell(r), 8)
v1 <- r[a[,1]]
v2 <- r[a[,2]]
m <- matrix(0, nrow=5,ncol=5)
for(i in 1:length(v1)) {m[v1[i], v2[i]] <- m[v1[i], v2[i]] +1}
m
```

---

adjacent *Adjacent cells*

Description

Identify cells that are adjacent to a set of cells on a raster. See adjacency for an alternative implementation.

Usage

```
adjacent(x, cells, directions=4, pairs=FALSE, target=NULL, sorted=FALSE, include=FALSE, id=FALSE)
```

Arguments

x	Raster* object
cells	vector of cell numbers for which adjacent cells should be found
directions	the number of directions in which cells should be connected: 4 (rook's case), 8 (queen's case), 16, or 'bishop'. Or a neigborhood matrix (see Details)
pairs	logical. If TRUE, a matrix of pairs of adjacent cells is returned. If FAlSE, a vector of cells adjacent to cells is returned

target	optional vector of target cell numbers that should be considered. All other adjacent cells are ignored
sorted	logical. Should the results be sorted?
include	logical. Should the focal cells be included in the result?
id	logical. Should the id of the cells be included in the result? (numbered from 1 to length(cells)

Details

A neighborhood matrix indentifies the cells around each cell that are considered adjacent. The matrix should have one, and only one, cell with value 0 (the focal cell); at least one cell with value 1 (the adjacent cell(s)); All other cells are not considered adjacent and ignored.

Cell numbers start with 1 in the upper-left corner and increase from left to right and from top to bottom.

Value

matrix or vector with adjacent cells.

Author(s)

Robert J. Hijmans and Jacob van Etten

See Also

adjacency

Examples

```
r <- raster(nrows=10, ncols=10)
adjacent(r, cells=c(1, 55), directions=8, pairs=TRUE)

a <- adjacent(r, cell = c(1,55,90), directions=4, sorted=TRUE)
a

r[c(1,55,90)] <- 1
r[a] <- 2
plot(r)

# same result as above
rook <- matrix(c(NA, 1, NA,
                  1, 0,  1,
                 NA, 1, NA), ncol=3, byrow=TRUE)

adjacent(r, cells = c(1,55,90), directions=rook, sorted=TRUE)
```

aggregate *Aggregate cells*

Description

Aggregate a Raster* object to create a new RasterLayer or RasterBrick with a lower resolution (larger cells). Aggregation groups rectangular areas to create larger cells. The value for the resulting cells is computed with a user-specified function.

Usage

```
## S4 method for signature 'Raster'
aggregate(x, fact=2, fun=mean, expand=TRUE, na.rm=TRUE, filename='', ...)
```

Arguments

x	Raster* object
fact	Integer. Aggregation factor expressed as number of cells in each direction (horizontally and vertically). Or two integers (horizontal and vertical aggregation factor). See Details
fun	Function used to aggregate values
expand	logical. If TRUE the output Raster* object will be larger then the input Raster* object if a division of the number of columns or rows with factor is not an integer
na.rm	logical. If TRUE, NA cells are removed from calculations
filename	Character. Output filename (optional)
...	Additional arguments as for writeRaster

Details

Aggregation will result in a Raster* object with fact*fact fewer cells; if necessary this number is adjusted according to the value of expand. For example, fact=2 will result in a new Raster* object with 2*2=4 times fewer cells. If two numbers are supplied, e.g., fact=c(2,3), the first will be used for aggregating in the horizontal direction, and the second for aggregating in the vertical direction, and the new RasterLayer will have 2*3=6 times fewer cells.

Aggregation starts at the upper-left end of a raster. If a division of the number of columns or rows with factor does not return an integer, the extent of the resulting Raster object will either be somewhat smaller or somewhat larger then the original RasterLayer. For example, if an input RasterLayer has 100 columns, and fact=12, the output Raster object will have either 8 columns (expand=FALSE) (using 8 x 12 = 96 of the original columns) or 9 columns (expand=TRUE). In both cases, the maximum x coordinate of the output RasterLayer would, of course, also be adjusted.

The function fun should take multiple numbers, and return a single number. For example mean, modal, min or max. It should also accept a na.rm argument (or ignore it as one of the 'dots' arguments).

Value

RasterLayer or RasterBrick

Author(s)

Robert J. Hijmans and Jacob van Etten

See Also

disaggregate, resample

Examples

```
r <- raster()
# a new aggregated raster, no values
ra <- aggregate(r, fact=10)
r <- setValues(r, runif(ncell(r)))

# a new aggregated raster, max of the values
ra <- aggregate(r, fact=10, fun=max)

# multiple layers
s <- stack(r, r*2)
x <- aggregate(s,2)
```

---

alignExtent *Align an extent (object of class Extent)*

Description

Align an Extent object with the (boundaries of the) cells of a Raster* object

Usage

```
alignExtent(extent, object, snap='near')
```

Arguments

extent	Extent object
object	Raster* object
snap	Character. One of 'near', 'in', or 'out', to determine in which direction the extent should be aligned. To the nearest border, inwards or outwards

Details

Aligning an Extent object to another object assures that it gets the same origin and resolution. This should only be used to adjust objects because of imprecision in the data. alignExtent should not be used to force data to match that really does not match (use e.g. resample or (dis)aggregate for this).

Value

Extent object

Author(s)

Robert J. Hijmans

See Also

extent, drawExtent, Extent-class

Examples

```
r <- raster()
e <- extent(-10.1, 9.9, -20.1, 19.9)
ea <- alignExtent(e, r)
e
extent(r)
ea
```

---

approxNA *Estimate values for cells that are NA*

Description

approxNA uses the stats function approx to estimate values for cells that are NA by interpolation across layers. Layers are considered equidistant, unless an argument 'z' is used, or getZ returns values, in which case these values are used to determine distance between layers.

For estimation based on neighboring cells see focal

Usage

```
approxNA(x, ...)
```

Arguments

x	RasterStack or RasterBrick object
...	additional arguments as in approxfun (except for x, y, which cannot be used) and an additional argument 'z' to indicated the distance between layers (e.g., time, depth)

Value

RasterBrick

Author(s)

Robert J. Hijmans

See Also

focal

Examples

```
r <- raster(ncols=5, nrows=5)
r1 <- setValues(r, runif(ncell(r)))
r2 <- setValues(r, runif(ncell(r)))
r3 <- setValues(r, runif(ncell(r)))
r4 <- setValues(r, runif(ncell(r)))
r5 <- setValues(r, NA)
r6 <- setValues(r, runif(ncell(r)))
r1[1:10] <- NA
r2[5:15] <- NA
r3[8:25] <- NA
s <- stack(r1,r2,r3,r4,r5,r6)
x1 <- approxNA(s)
x2 <- approxNA(s, rule=2)

x3 <- approxNA(s, rule=2, z=c(1,2,3,5,14,15))
```

---

area *Size of cells*

Description

Compute the approximate surface area of cells in an unprojected (longitude/latitude) Raster object. It is an approximation because area is computed as the height (latitudial span) of a cell (which is constant among all cells) times the width (longitudinal span) in the (latitudinal) middle of a cell. The width is smaller at the poleward side than at the equator-ward side of a cell. This variation is greatest near the poles and the values are thus not very precise for very high latitudes.

Usage

```
## S4 method for signature 'RasterLayer'
area(x, filename="", na.rm=FALSE, weights=FALSE, ...)

## S4 method for signature 'RasterStackBrick'
area(x, filename="", na.rm=FALSE, weights=FALSE, ...)
```

Arguments

x	Raster* object
filename	Character. Filename for the output Raster object (optional)
na.rm	Logical. If TRUE, cells that are NA are ignored
weights	Logical. If TRUE, the area of each cells is divided by the total area of all cells that are not NA
...	Additional arguments as for writeRaster

Details

If x is a RasterStack/Brick, a RasterBrick will be returned if na.rm=TRUE. However, if na.rm=FALSE, a RasterLayer is returned, because the values would be the same for all layers.

Value

RasterLayer or RasterBrick. Cell values represent the size of the cell in km2, or the relative size if weights=TRUE

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(nrow=18, ncol=36)
a <- area(r)
```

Arith-methods *Arithmetic with Raster* objects*

Description

Standard arithmetic operators for computations with Raster* objects and numeric values. The following operators are available: `+, -, *, /, ^, %%, %/%`

Input Raster* objects should have the same extent, origin and resolution. If only the extent differs, the computation will continue for the intersection of the Raster objects. Operators are applied on a cell by cell basis. For a RasterLayer, numeric values are recycled by row. For a RasterStack or RasterBrick, recycling is done by layer. RasterLayer objects can be combined RasterStack/Brick objects, in which case the RasterLayer is 'recycled'. When using multiple RasterStack or RasterBrick objects, the number of layers of these objects needs to be the same.

Details

If the values of the output Raster* cannot be held in memory, they will be saved to a temporary file. You can use options to set the default file format, datatype and progress bar.

Value

A Raster* object, and in some cases the side effect of a new file on disk.

Author(s)

Robert J. Hijmans

See Also

Math-methods, overlay, calc

Examples

```
r1 <- raster(ncols=10, nrows=10)
r1[] <- runif(ncell(r1))
r2 <- setValues(r1, 1:ncell(r1) / ncell(r1) )
r3 <- r1 + r2
r2 <- r1 / 10
r3 <- r1 * (r2 - 1 + r1^2 / r2)

# recycling by row
r4 <- r1 * 0 + 1:ncol(r1)

# multi-layer object mutiplication, no recycling
b1 <- brick(r1, r2, r3)
b2 <- b1 * 10

# recycling by layer
b3 <- b1 + c(1, 5, 10)

# adding twoh RasterBrick objects
b3 <- b2 + b1

# summing two RasterBricks and one RasterLayer. The RasterLayer is 'recycled'
b3 <- b1 + b2 + r1
```

---

as.data.frame *Get a data.frame with raster cell values*

Description

as.matrix returns all values of a Raster* object as a matrix. For RasterLayers, rows and columns in the matrix represent rows and columns in the RasterLayer object. For other Raster* objects, the matrix returned by as.matrix has columns for each layer and rows for each cell.

as.array returns an array of matrices that are like those returned by as.matrix for a RasterLayer

If there is insufficient memory to load all values, you can use getValues or getValuesBlock to read chunks of the file.

Usage

```
as.data.frame(x, row.names=NULL, optional=FALSE, ...)
```

Arguments

x	Raster* object
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional.
...	Additional arguments (none).

Value

data.frame

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(ncol=3, nrow=3)
r[] = 1:ncell(r)
as.data.frame(r)
s <- stack(r,r)
as.data.frame(s)
```

---

as.logical *Change values to logical*

Description

Change values of a Raster* object to logical values (zero becomes FALSE, all other values become TRUE) You can provide the standard additional arguments: filename, format, overwrite, and progress.

See Also

as.logical

Examples

```
r <- raster(nrow=10, ncol=10)
r[] <- round(runif(ncell(r)))
r <- as.logical(r)
```

`as.matrix` *Get a matrix with raster cell values*

Description

`as.matrix` returns all values of a Raster* object as a matrix. For RasterLayers, rows and columns in the matrix represent rows and columns in the RasterLayer object. For other Raster* objects, the matrix returned by `as.matrix` has columns for each layer and rows for each cell.

`as.array` returns an array of matrices that are like those returned by `as.matrix` for a RasterLayer

If there is insufficient memory to load all values, you can use `getValues` or `getValuesBlock` to read chunks of the file.

Usage

```
as.matrix(x, ...)
as.array(x, ...)
```

Arguments

`x`	Raster* object
`...`	Additional arguments.
	`maxpixels` Integer. To regularly subsample very large objects
	`transpose` Logical. Transpose the data? (for as.array only)

Value

matrix or array

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(ncol=3, nrow=3)
r[] = 1:ncell(r)
as.matrix(r)
s <- stack(r,r)
as.array(s)
```

as.raster *Coerce to a 'raster' object*

Description

Implementation of the generic as.raster function to create a 'raster' (small r) object. NOT TO BE CONFUSED with the Raster* (big R) objects defined by the raster package! Such objects can be used for plotting with the rasterImage function.

Usage

```
as.raster(x, ...)
```

Arguments

x	RasterLayer object
...	Additional arguments.
	maxpixels Integer. To regularly subsample very large objects
	col Vector of colors. Default is col=rev(terrain.colors(255)))

Value

'raster' object

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(ncol=3, nrow=3)
r[] <- 1:ncell(r)
as.raster(r)
```

atan2 *Two argument arc-tangent*

Description

For RasterLayer arguments x and y, atan2(y, x) returns the angle in radians for the tangent y/x, handling the case when x is zero. See link[base]{Trig}

See Math-methods for other trigonometric and mathematical functions that can be used with Raster* objects.

Usage

```
atan2(y, x)
```

Arguments

y	RasterLayer object
x	RasterLayer object

Author(s)

Robert J. Hijmans

See Also

Math-methods

Examples

```
r1 <- r2 <- raster(nrow=10, ncol=10)
r1[] <- (runif(ncell(r1))-0.5) * 10
r2[] <- (runif(ncell(r1))-0.5) * 10
atan2(r1, r2)
```

---

autocorrelation *Spatial autocorrelation*

Description

Compute Moran's I or Geary's C measuers of global spatial autocorrelation in a RasterLayer, or compute the the local Moran or Geary index (Anselin, 1995).

Usage

```
Geary(x, w=3)
Moran(x, w=3)
MoranLocal(x, w=3)
GearyLocal(x, w=3)
```

Arguments

x	RasterLayer
w	Spatial weights. Either a single number or a vector of two numbers to define a neighborhood (as in focal) or a rectangular matrix with uneven sides

Details

The default setting uses a 3x3 neighborhoud to compute "Queen's case" indices. You can use a filter (weights matrix) to do other things, such as "Rook's case", or different lags.

Value

A single value (Moran's I or Geary's C) or a RasterLayer (Local Moran or Geary values)

Author(s)

Robert J. Hijmans and Babak Naimi

References

Moran, P.A.P., 1950. Notes on continuous stochastic phenomena. Biometrika 37:17-23

Geary, R.C., 1954. The contiguity ratio and statistical mapping. The Incorporated Statistician 5: 115-145

Anselin, L., 1995. Local indicators of spatial association-LISA. Geographical Analysis 27:93-115

http://en.wikipedia.org/wiki/Indicators_of_spatial_association

See Also

The spdep package for additional and more general approaches for computing indices of spatial autocorrelation

Examples

```
r <- raster(nrows=10, ncols=10)
r[] <- 1:ncell(r)

Moran(r)
# Rook's case
f <- matrix(c(0,1,0,1,0,1,0,1,0), nrow=3)
Moran(r, f)

Geary(r)

x1 <- MoranLocal(r)

# Rook's case
x2 <- MoranLocal(r, w=f)
```

---

bands *Number of bands*

Description

A 'band' refers to a single layer for a possibly multi-layer file. Most RasterLayer objects will refer to files with a single band. (The term 'band' is frequently used in remote sensing to refer to a variable (layer) in a multi-variable dataset and in that context bands could be stored in a single or in seperate files).

nbands returns the number of bands of the file that a RasterLayer points to (and 1 if it does not point at any file). This functions also works for a RasterStack for which it is equivalent to nlayers.

band returns the specific band the RasterLayer refers to (1 if the RasterLayer points at single layer file or does not point at any file).

Usage

```
nbands(x)
bandnr(x, ...)
```

Arguments

x	RasterLayer object
...	Additional arguments (none at this time)

Value

a numeric value >= 1

Author(s)

Robert J. Hijmans

See Also

nlayers

Examples

```
r <- raster()
nbands(r)
bandnr(r)
```

---

blockSize *Block size for writing files*

Description

This function can be used to suggest a chunk size, and corresponding row numbers, to be used when processing Raster objects chunk by chunk, normally together with writeValues.

Usage

```
blockSize(x, chunksize, n=nlayers(x), minblocks=4, minrows=1)
```

Arguments

x	Raster* object
chunksize	Integer, normally missing. Can be used to set the block size; unit is number of cells. Block size is then computed in units of number of rows (always >= 1)
n	Integer. number of layers to consider. The function divides chunksize by n to determine blocksize
minblocks	Integer. Minimum number of blocks
minrows	Integer. Minimum number of rows in each block

Value

A list with rows, the suggested row numbers at which to start the blocks for reading and writing, size, the block size (number of rows) and n, the total number of blocks

Author(s)

Robert J. Hijmans

See Also

writeValues

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
blockSize(r)
```

---

boxplot *Box plot of Raster objects*

Description

Box plot of layers in a Raster object

Usage

```
boxplot(x, ...)
```

Arguments

x	Raster* object
...	Additional arguments. See Methods

Methods

```
boxplot(x, maxpixels=100000, ...)
```

maxpixels	Integer. Number of pixels to sample from each layer of large Raster objects
...	Argument passed to graphics::boxplot

Author(s)

Robert J. Hijmans

See Also

pairs, hist

Examples

```
r1 <- r2 <- r3 <- raster(ncol=10, nrow=10)
r1[] <- rnorm(ncell(r1), 100, 40)
r2[] <- rnorm(ncell(r1), 80, 10)
r3[] <- rnorm(ncell(r1), 120, 30)
s <- stack(r1, r2, r3)
layerNames(s) <- c('A', 'B', 'C')

boxplot(s, notch=TRUE, col=c('red', 'blue', 'orange'), main='Box plot', ylab='random' )
```

---

brick *Create a RasterBrick object*

Description

A RasterBrick is a multi-layer raster object. They are typically created from a multi-band file; but they can also exist entirely in memory. They are similar to a RasterStack, but processing time should be shorter when using a RasterBrick. Yet they are less flexible as they can only point to a single file.

A RasterBrick can be created from RasterLayer objects, from a RasterStack, or from a (multi-band) file. The can also be created from SpatialPixels*, SpatialGrid*, and Extent objects, and from an three-dimensional array.

Usage

```
brick(x, ...)
```

Arguments

x	Filename (character), Raster* object, array, SpatialGrid*, SpatialPixels*, Extent, or list of Raster* objects
...	Additional arguments. See under Details

Value

RasterBrick object

Methods

1) Create a RasterBrick from a file

```
brick(x, ...)
```

x	character. A filename of a file that the raster package can read. Supported file types are the 'native' raster package fo
...	additional arguments.

Additional arguments:

native	Logical. Default is FALSE except when package rgdal is missing. If TRUE, reading and writing of IDRISI, BIL, B

For netCDF files (CF convention):

varname	character. The variable name (e.g. 'altitude' or 'precipitation'. If not supplied and the file has multiple variables
level	integer > 0 (default=3). To select the 'level' (4th dimension variable) to use, if the data has 4 dimensions

To read netCDF files, the ncdf package needs to be available.

2) Create a RasterBrick from scratch

```
brick(nrows=180, ncols=360, xmn=-180, xmx=180, ymn=-90, ymx=90, crs="+proj=longlat
+datum=WGS84", nl=1)
```

nrows	number of rows
ncols	number of columns
xmn	minimum x coordinate (left border)
xmx	maximum x coordinate (right border)
ymn	minimum y coordinate (bottom border)
ymx	maximum y coordinate (top border)
crs	Character or object of class CRS. PROJ4 type description of a Coordinate Reference System (map projection)
nl	the desired number of layers

(item x is 'missing' in this case)

3) Create a RasterBrick from an Extent object

```
brick(x, nrows=10, ncols=10, crs=NA, nl=1)
```

x	Extent object
nrows	number of rows
ncols	number of columns
crs	Character or object of class CRS. PROJ4 type description of a Coordinate Reference System (map projection)
nl	the desired number of layers

4) Create a RasterBrick from a RasterStack

```
brick(x, values=TRUE, nl)
```

x	a RasterStack object
values	Logical. If TRUE, the values of the RasterStack are transferred to the RasterBrick
nl	the desired number of layers (optional. If set, values becomes FALSE

5) Create a RasterBrick from a RasterLayer or RasterBrick

```
brick(x, nl=1)
```

This copies the parameters of a RasterStack object to a new RasterBrick. The values of the RasterStack are transferred to the RasterBrick, unless values=FALSE.

x	a Raster* object
nl	the desired number of layers

6) Create a RasterBrick from a SpatialPixels* or SpatialGrid*

```
brick(x, layer=0)
```

7) Create a RasterBrick from an array

The default extent is set to be between 0 and 1 in the x and y direction but can be changed at creation of the RasterLayer object or later. You can also provide a projection.

```
function(x, xmn=0, xmx=1, ymn=0, ymx=1, crs=NA, transpose=FALSE)
```

x	three-dimensional array (rows, columns, layers)
xmn	minimum x coordinate (left border)
xmx	maximum x coordinate (right border)
ymn	minimum y coordinate (bottom border)
ymx	maximum y coordinate (top border)
crs	PROJ4 type description of a map projection (optional)
transpose	Logical. Transpose the data?

Author(s)

Robert J. Hijmans

See Also

raster

Examples

```
b <- brick(system.file("external/rlogo.grd", package="raster"))
b
nlayers(b)
layerNames(b)
extract(b, 870)
```

buffer *buffer*

Description

Calculate a buffer around all cells that are not NA.

Note that the distance unit of the buffer `width` parameter is meters if the RasterLayer is not projected (`+proj=longlat`), and in map units (typically also meters) when it is projected.

Usage

```
## S4 method for signature 'RasterLayer'
buffer(x, width=0, filename='', doEdge=FALSE, ...)
```

Arguments

x	RasterLayer object
width	Numeric. Number > 0. Unit is meter if x has a longitude/latitude CRS, or mapunits in other cases
filename	Character. Filename for the output RasterLayer (optional)
doEdge	Logical. If TRUE, the edge function is called first. This may be efficient in cases where you compute a buffer around very large areas. Calling edge determines the edge cells that matter for distance computation
...	Additional arguments as for writeRaster

Value

RasterLayer

Author(s)

Robert J. Hijmans

See Also

distance, gridDistance, pointDistance

Examples

```
r <- raster(ncol=36,nrow=18)
r[] <- NA
r[500] <- 1
b <- buffer(r, width=5000000)
#plot(b)
```

`calc` *Calculate*

Description

Calculate values for a new Raster* object from another Raster* object, using a formula.

If `x` is a RasterLayer, `fun` is typically a function that can take a single vector as input, and return a vector of values of the same lenght (e.g. `sqrt`). If `x` is a RasterStack or RasterBrick, fun should operate on a vector of values (one vector for each cell). `calc` returns a RasterLayer if `fun` returns a single value (e.g. `sum`) and it returns a RasterBrick if `fun` returns more than one number, e.g., `fun=quantile`.

In many cases, what can be achieved with `calc` in a more intuitive 'raster-algebra' notation (see `Arith-methods`). For example, `r <- r * 2` rather than `r <- calc(r, fun=function(x){x * 2}`, or `r <- sum(s)` rather than `r <- calc(s, fun=sum)`. However, `calc` should be faster when using complex formulas on large datasets. With `calc` it is possible to set an output filename and file type preferences.

See (`overlay`) to use functions that refer to specific layers, like (`function(a,b,c){a + sqrt(b) / c}`)

Usage

```
## S4 method for signature 'Raster,function'
calc(x, fun, filename='', na.rm, forcefun=FALSE, forceapply=FALSE, ...)
```

Arguments

`x`	Raster* object
`fun`	function
`filename`	character. Output filename (optional)
`na.rm`	Remove `NA` values, if supported by 'fun' (only relevant when summarizing a multilayer Raster object into a RasterLayer)
`forcefun`	logical. For debugging. Force `calc` to not use fun with apply
`forceapply`	logical. For debugging. Force `calc` to use fun with apply
`...`	Additional arguments as for `writeRaster`

Value

a Raster* object

Note

For large objects `calc` will do computate values chunk by chunk. This means that for the result of `fun` to be correct it should not depend on having access to _all_ values at once. For example, to scale the values of a Raster* object by subtracting its mean value (for each layer), you would _not_ do, for Raster object `x`:

```
calc(x, function(x)scale(x, scale=FALSE))
```

Because the mean value of each chunk will likely be different. Rather do something like

```
m <- cellStats(x, 'mean')
x - m
```

Author(s)

Robert J. Hijmans and Matteo Mattiuzzi

See Also

`overlay`, `reclass`, Arith-methods, Math-methods, `reclass`

Examples

```
r <- raster(ncols=36, nrows=18)
r[] <- 1:ncell(r)

# multiply values with 10
fun <- function(x) { x * 10 }
rc1 <- calc(r, fun)

# set values below 100 to NA.
fun <- function(x) { x[x<100] <- NA; return(x) }
rc2 <- calc(r, fun)

# set NA values to -9999
fun <- function(x) { x[is.na(x)] <- -9999; return(x)}
rc3 <- calc(rc2, fun)

# using a RasterStack as input
s <- stack(r, r*2, sqrt(r))
# return a RasterLayer
rs1 <- calc(s, sum)

# return a RasterBrick
rs2 <- calc(s, fun=function(x){x * 10})
# recycling by layer
rs3 <- calc(s, fun=function(x){x * c(1, 5, 10)})

# use overlay when you want to refer to indiviudal layer in the function
# but it can be done with calc:
rs4 <- calc(s, fun=function(x){x[1]+x[2]*x[3]})

##
```

```
# Some regression examples
##

# create data
r <- raster(nrow=10, ncol=10)
s1 <- s2<- list()
for (i in 1:12) {
s1[i] <- setValues(r, rnorm(ncell(r), i, 3) )
s2[i] <- setValues(r, rnorm(ncell(r), i, 3) )
}
s1 <- stack(s1)
s2 <- stack(s2)

# regression of values in one brick (or stack) with another
s <- stack(s1, s2)
# s1 and s2 have 12 layers; coefficients[2] is the slope
fun <- function(x) { lm(x[1:12] ~ x[13:24])$coefficients[2] }
x1 <- calc(s, fun)

# regression of values in one brick (or stack) with 'time'
time <- 1:nlayers(s)
fun <- function(x) { lm(x ~ time)$coefficients[2] }
x2 <- calc(s, fun)

# get multiple layers, e.g. the slope _and_ intercept
fun <- function(x) { lm(x ~ time)$coefficients }
x3 <- calc(s, fun)
```

`cellFrom` *Get cell, row, or column number*

Description

Get cell number(s) of a Raster* object from row and/or column numbers. Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom. The last cell number equals the number of cells of the Raster* object.

Usage

```
cellFromRowCol(object, rownr, colnr)
cellFromRowColCombine(object, rownr, colnr)
cellFromRow(object, rownr)
cellFromCol(object, colnr)
colFromX(object, x)
rowFromY(object, y)
cellFromXY(object, xy)
cellFromLine(object, lns)
cellFromPolygon(object, p, weights=FALSE)
```

Arguments

object	Raster* object (or a SpatialPixels* or SpatialGrid* object)
colnr	column number; or vector of column numbers
rownr	row number; or vector of row numbers
x	x coordinate(s)
y	y coordinate(s)
xy	matrix of x and y coordinates, or a SpatialPoints or SpatialPointsDataFrame object
lns	SpatialLines object
p	SpatialPolygons object
weights	Logical. If TRUE, the fraction of each cell that is covered is also returned

Details

cellFromRowCol returns the cell numbers obtained for each row / col number pair. In contrast, cellFromRowColCombine returns the cell numbers obtained by the combination of all row and column numbers supplied as arguments.

Value

row, column or cell number(s). cellFromLine and cellFromPolygon return a list.

Author(s)

Robert J. Hijmans

See Also

xyFromCell, cellsFromExtent

Examples

```
r <- raster(ncols=10, nrows=10)
cellFromRowCol(r, 5, 5)
cellFromRowCol(r, 1:2, 1:2)
cellFromRowColCombine(r, 1:3, 1:2)
cellFromCol(r, 1)
cellFromRow(r, 1)
colFromX(r, 0.5)
rowFromY(r, 0.5)
cellFromXY(r, c(0.5, 0.5))

cds1 <- rbind(c(-180,-20), c(-160,5), c(-60, 0), c(-160,-60), c(-180,-20))
cds2 <- rbind(c(80,0), c(100,60), c(120,0), c(120,-55), c(80,0))
pols <- SpatialPolygons(list(Polygons(list(Polygon(cds1)), 1), Polygons(list(Polygon(cds2)), 2)))
cellFromPolygon(r, pols)
```

cellsFromExtent *Cells from Extent*

Description

This function returns the cell numbers for a Raster* object that are within a specfied extent (rectangular area), supply either an object of class Extent, or another Raster* object.

Usage

```
cellsFromExtent(object, extent, expand=FALSE)
```

Arguments

object	A Raster* object
extent	An object of class Extent (which you can create with newExtent(), or another Raster* object)
expand	Logical. If TRUE, NA is returned for (virtual) cells implied by bndbox, that are outside the RasterLayer (object). If FALSE, only cell numbers for the area where object and bndbox overlap are returned (see intersect)

Value

a vector of cell numbers

Author(s)

Robert J. Hijmans

See Also

extent, cellFromXY

Examples

```
r <- raster()
bb <- extent(-5, 5, -5, 5)
cells <- cellsFromExtent(r, bb)
r <- crop(r, bb)
r[] <- cells
```

`cellStats` *Cell statistics*

Description

Compute statistics for the cells of each layer of a Raster object. In the `raster` package, functions such as max, min, and mean, when used with Raster objects as argument, return a new Raster object (with a value computed for each cell). In contrast, cellStats returns a single value, computed from the all the values of a layer.

Usage

```
cellStats(x, stat='mean', ...)
```

Arguments

`x`	A Raster* object
`stat`	The function to be applied
`...`	Additional arguments

Details

`cellStats` will fail (gracefully) for very large Raster objects except for a number of known functions: sum, mean, min, max, sd, 'countNA'. 'countNA' must be supplied as a character value (with quotes), the other known functions may be supplied with or without quotes. For other functions you could perhaps use a sample of the RasterLayer that can be held in memory (see `sampleRandom` and `sampleRegular`)

Value

Numeric.

Author(s)

Robert J. Hijmans

See Also

`quantile` , `minValue` , `maxValue` , `setMinMax`

Examples

```
r <- raster(nrow=18, ncol=36)
r[] <- runif(ncell(r)) * 10
# works for large files
cellStats(r, 'mean')
# same, but does not work for very large files
cellStats(r, mean)
```

```
# multi-layer object
cellStats(brick(r,r), mean)
```

`clearValues` *Clear values*

Description

Clear cell values of a Raster* object from memory

Usage

```
clearValues(x)
```

Arguments

x	Raster* object

Value

a Raster* object

Author(s)

Robert J. Hijmans

See Also

values, replacement

Examples

```
r <- raster(ncol=10, nrow=10)
r[] <- 1:ncell(r)
r <- clearValues(r)
```

click *Click on a map*

Description

Click on a map (plot) to get values of a Raster* or SpatialGrid or SpatialPixels object at that location; and optionally the coordinates and cell number of the location.

Usage

```
## S4 method for signature 'Raster'
click(x, n=1, id=FALSE, xy=FALSE, cell=FALSE, type = "n", ...)

## S4 method for signature 'SpatialGrid'
click(x, n=1, id=FALSE, xy=FALSE, type = "n", ...)
```

Arguments

x	Raster*, or Spatial* object (or missing)
n	number of clicks on the map
id	Logical. If TRUE, a numeric ID is shown on the map that corresponds to the row number of the output
xy	Logical. If TRUE, xy coordinates are included in the output
cell	Logical. If TRUE, cell numbers are included in the output
type	One of "n", "p", "l" or "o". If "p" or "o" the points are plotted; if "l" or "o" they are joined by lines. See ?locator
...	additional graphics parameters used if type != "n" for plotting the locations. See ?locator

Value

The value(s) of object at the point(s) clicked on; and the coordinates of the center of the cell if xy==TRUE.

Note

The plot only provides the coordinates (xy pair), the values are read from the Raster* object that is passed as an argument. Thus you can extract values for a Raster* object that is not plotted, as long as it shares the coordinate system (CRS) with the layer that is plotted.

Unless the process is terminated prematurely values at at most n positions are determined. The identification process can be terminated by clicking the second mouse button and selecting 'Stop' from the menu, or from the 'Stop' menu on the graphics window.

Author(s)

Robert J. Hijmans

See Also

select, drawExtent

Examples

```
 r <- raster(system.file("external/test.grd", package="raster"))
#plot(r)
#click(r)
#now click on the plot (map)
```

---

clump *Detect clumps*

Description

Detect clumps (patches) of connected cells. Each clump gets a unique ID. NA and zero are used as background values (i.e. these values are used to seperate clumps). You can use queen's or king's case, using the directions argument. For larger files that are processed in chunks, the highest clump number is not necessarily equal to the number of clumps (unless you use argument gaps=FALSE).

Usage

```
## S4 method for signature 'RasterLayer'
clump(x, filename="", directions=8, gaps=TRUE, ...)
```

Arguments

x	RasterLayer object
filename	Character. Filename for the output RasterLayer (optional)
directions	Integer. Which cells are considered adjacent? Should be 8 (Queen's case) or 4 (Rook's case)
gaps	Logical. If TRUE (the default), there may be 'gaps' in the chunk numbers (e.g. you may have clumps with IDs 1, 2, 3 and 5, but not 4). If it is FALSE, these numbers will be recoded from 1 to n (4 in this example)
...	Additional arguments as for writeRaster

Value

RasterLayer

Note

This function requires that the igraph package is available.

Author(s)

Robert J. Hijmans and Jacob van Etten

Examples

```
r <- raster(ncols=12, nrows=12)
r[] <- round(runif(ncell(r))*0.6 )
rc <- clump(r)
freq(rc)
#plot(rc)
```

---

cluster *Use a multi-core cluster*

Description

`beginCluster` creates, and `endCluster` deletes a 'snow' cluster object. This object can be used for multi-core computing with those 'raster' functions that support it.

`beginCluster` determines the number of nodes (cores) that are available and uses all of them (unless the argument `n` is used).

NOTE: beginCluster may fail when the package 'nws' is installed. You can fix that by removing the 'nws' package, or by setting the cluster type manually, e.g. `beginCluster(type="SOCK")`

endCluster closes the cluster and removes the object.

The use of the cluster is automatic in these functions" `predict`, `projectRaster`, `resample` and in `extract` when using polygons.

`clusterR` is a flexible interface for using cluster with other functions. This function only works with functions from the raster package that return Raster* objects that operate on a cell by cell basis and return an object with the same number of cells as the input raster object(s). Among other functions, it does _not_ work with mrege, crop, mosaic, (dis)aggregate, resample, projectRaster, focal, distance, direction.

Usage

```
beginCluster(n, type, nice, exclude)
endCluster()
clusterR(x, fun, args=NULL, filename='', cl=NULL, m=2, ...)
```

Arguments

`n`	Integer. The number of nodes to be used (optional)
`type`	Character. The cluster type to be used (optional). E.g. "SOCK", see the snow package for details
`nice`	Integer. To set the prioirty for the workers, between -20 and 20 (UNIX like platforms only)

exclude	Character. Packages to exclude from loading on the nodes (because they may fail there) but are required/loaded on the master
x	Raster* object
fun	function
args	arguments for the function (excluding x, which should always be the first argument
filename	character. Output filename (optional)
cl	cluster object (do not use it if beginCluster() has been called
m	tuning parameter to determine how many blocks should be used. The number is rounded and multiplied with the number of nodes.
...	additional arguments for writing Raster objects to disk

Value

None. The side effect is to create or delete a cluster object.

Note

If you want to write your own cluster-enabled functions see getCluster, returnCluster, and the vignette about writing functions

Author(s)

Matteo Mattiuzzi and Robert J. Hijmans

Examples

```
## Not run:

#beginCluster()
beginCluster(type="SOCK")

r <- raster()
r[] <- 1:ncell(r)

x <- clusterR(r, sqrt, verbose=T)

f1 <- function(x) calc(x, sqrt)
y <- clusterR(r, f1)

s <- stack(r, r*2, r*3)
f2 <- function(x) calc(x, range)
z <- clusterR(s, f2)

pts <- matrix(c(0,0, 45,45), ncol=2, byrow=T)
d <- clusterR(r, distanceFromPoints, args=list(xy=pts))

endCluster()
```

```
## End(Not run)
```

---

compare *Compare*

Description

Evaluate whether a two or more rasters have the same extent, number of rows and columns, projection, resolution, and origin (or a subset of these comparisons). Cell values are not compared by this function.

Usage

```
compare(x, ..., extent=TRUE, rowcol=TRUE, crs=TRUE, res=FALSE, orig=FALSE,
rotation=TRUE, tolerance, stopiffalse=TRUE, showwarning=FALSE)
```

Arguments

x	A Raster* object
...	Additional Raster* objects
extent	Logical. If TRUE, bounding boxes are compared
rowcol	Logical. If TRUE, number of rows and columns of the objects are compared
crs	Logical. If TRUE, coordinate reference systems are compared.
res	Logical. If TRUE, resolutions are compared (redundant when checking extent and rowcol)
orig	Logical. If TRUE, origins are compared
rotation	Logical. If TRUE, rotations are compared
tolerance	Numeric value between 0 and 0.5. If not supplied, the default value is used (see showOptions. It sets difference (relative to the cell resolution) that is permissable for objects to be considered 'equal', if they have a non-integer origin or resolution. See all.equal.
stopiffalse	Logical. If TRUE, an error will occur if the objects are not the same
showwarning	Logical. If TRUE, an warning will be given if objects are not the same. Only relevant when stopiffalse is TRUE

Author(s)

Robert J. Hijmans

Examples

```
r1 <- raster()
r2 <- r1
r3 <- r1
compare(r1, r2, r3)
nrow(r3) <- 10
# compare(r1, r3)
compare(r1, r3, stopiffalse=FALSE)
compare(r1, r3, rowcol=FALSE)
```

Compare-methods *Compare Raster* objects*

Description

These methods compare the location and resolution of Raster* objects. That is, they compare their spatial extent, projection, and number of rows and columns.

For BasicRaster objects you can use == and !=, the values returned is a single logical value TRUE or FALSE

For RasterLayer objects, they also compare the values associated with the objects, and the result is a logical (Boolean) RasterLayer object. And the value returned is a RasterLayer object, if the location and resolution of the two RasterLayers match, or an error of they do not match.

The following methods have been implemented for RasterLayer objects:

==, !=, >, <, <=, >=

Value

A logical value or a RasterLayer object, and in some cases the side effect of a new file on disk.

Author(s)

Robert J. Hijmans

Examples

```
r1 <- raster()
r1 <- setValues(r1, round(10 * runif(ncell(r1))))
r2 <- setValues(r1, round(10 * runif(ncell(r1))))
as(r1, 'BasicRaster') == as(r2, 'BasicRaster')
r3 <- r1 == r2

b <- extent(0, 360, 0, 180)
r4 <- setExtent(r2, b)
as(r2, 'BasicRaster') != as(r4, 'BasicRaster')
# The following would give an error. You cannot compare RasterLayer
# that do not have the same BasicRaster properties.
#r3 <- r1 > r4
```

`contour` *Contour plot*

Description

Contour plot of a RasterLayer. This is a generic function, in this package implemented for RasterLayer objects.

Usage

```
contour(x, ...)
```

Arguments

`x`	A Raster* object
`...`	Any argument that can be passed to `contour` (graphics package)

Methods

```
contour(x, y=1, maxpixels=100000, ...)
```

`x`	RasterLayer object
`y`	The layer number (integer > 0) if x is a RasterStack or RasterBrick
`maxpixels`	Maximum number of pixels used to create the contours
`...`	Any argument that can be passed to `contour` (graphics package)

Author(s)

Robert J. Hijmans

See Also

`persp`, `filledContour`

The rasterVis package has more advanced plotting methods for Raster* objects.

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
plot(r)
contour(r, add=TRUE)
```

count *Count*

Description

Count the frequency of a single value in a Raster object.

Usage

```
count(x, value, ...)
```

Arguments

x	A Raster* object
value	The value to be counted, can be a number or a logical or NA
...	Additional arugments: digits, the number of digits for rounding the values, default is 0, i.e. numbers are compared as integers.

Value

numeric

Author(s)

Robert J. Hijmans

See Also

freq

Examples

```
r <- raster(nrow=18, ncol=36)
r[] <- runif(ncell(r))
r <- r * r * r * 10
count(r, 5)
```

cover — *Replace NA values with values of other layers*

Description

Replace NA values in the first Raster object (x) with the values of the second (y), and so forth for additional Rasters. If x has mutliple layers, the subsequent Raster objects should have the same number of layers, or have a single layer only (which will be recycled).

Usage

```
cover(x, y, ...)
```

Arguments

x	Raster* object
y	Raster* object
...	Additional Raster objects, and additional arguments as for writeRaster

Value

RasterLayer or RasterBrick object

Author(s)

Robert J. Hijmans

Examples

```
r1 <- raster(ncols=36, nrows=18)
r1[] <- 1:ncell(r1)
r2 <- setValues(r1, runif(ncell(r1)))
r2[r2<0.5] <- NA
r3 <- cover(r2, r1)
```

crop — *Crop*

Description

crop returns a geographic subset of an object as specified by an Extent object (or object from which an extent object can be extracted/created). If x is a Raster* object, the Extent is aligned to x. Areas included in y but outside the extent of x are ignored (see expand if you want a larger area)

Usage

```
## S4 method for signature 'Raster'
crop(x, y, filename="", snap='near', ...)

## S4 method for signature 'Spatial'
crop(x, y, ...)
```

Arguments

x	Raster* object or SpatialPolygons* or SpatialLines* object
y	Extent object, or any object from which an Extent object can be extracted (see Details)
filename	Character, output filename. Optional
snap	Character. One of 'near', 'in', or 'out', for use with alignExtent
...	Additional arguments as for writeRaster

Details

Objects from which an Extent can be extracted/created include RasterLayer, RasterStack, RasterBrick and objects of the Spatial* classes from the sp package. You can check this with the extent function. New Extent objects can be also be created with function extent and drawExtent by clicking twice on a plot.

Value

RasterLayer or RasterBrick object; or SpatialLines or SpatialPolygons object.

Author(s)

Robert J. Hijmans

See Also

expand, merge

Examples

```
r <- raster(nrow=45, ncol=90)
r[] <- 1:ncell(r)
e <- extent(-160, 10, 30, 60)
rc <- crop(r, e)
```

crosstab *Cross-tabulate*

Description

Cross-tabulate two RasterLayer objects to create a contingency table.

Usage

```
crosstab(x, y, ...)
```

Arguments

x	RasterLayer object
y	RasterLayer object
...	Additional arguments. See Details

Details

A full call to the crosstab method for a RasterLayer is:

```
crosstab(x, y, digits=0, long=FALSE, progress)
```

digits	Integer. The number of digits for rounding the values before cross-tabulation
long	Logical. If TRUE the results are returned in 'long' format (matrix with three columns) instead of a table
progress	Character. "text", "window", or "" (the default, no progress bar)
...	Additional arguments that can be passed on to table, such as exclude=NULL to include NA values in the tabula

Value

A table or matrix

Author(s)

Robert J. Hijmans

See Also

freq, zonal

Examples

```
r <- raster()
r[] = runif(ncell(r)) * 5
s = setValues(r, runif(ncell(r)) * 10)
crosstab(r,s)
```

cut *Convert values to classes*

Description

Cut uses the base function cut to classify the values of a Raster* object according to which interval they fall in. The intervals are defined by the argument breaks. The leftmost interval corresponds to level one, the next leftmost to level two and so on.

Usage

```
cut(x, ...)
```

Arguments

x	A Raster* object
...	additional arguments. See cut

Value

Raster* object

Author(s)

Robert J. Hijmans

See Also

subs, reclass, calc

Examples

```
r <- raster(ncols=36, nrows=18)
r[] <- rnorm(ncell(r))
breaks <- -2:2 * 3
rc <- cut(r, breaks=breaks)
```

cv *Coefficient of variation*

Description

Compute the coefficient of variation (expressed as a percentage). If there is only a single value, sd is NA and cv returns NA if aszero=FALSE (the default). However, if (aszero=TRUE), cv returns 0.

Usage

```
## S4 method for signature 'ANY'
cv(x, ..., aszero=FALSE, na.rm = FALSE)

## S4 method for signature 'Raster'
cv(x, ..., aszero=FALSE, na.rm = FALSE)
```

Arguments

x	A vector of numbers (typically integers for modal), or a Raster* object
...	additional (vectors of) numbers, or Raster objects
aszero	logical. If TRUE, a zero is returned (rather than an NA) if the cv of single value is computed
na.rm	Remove (ignore) NA values

Value

vector or RasterLayer

Author(s)

Robert J. Hijmans

Examples

```
data <- c(0,1,2,3,3,3,3,4,4,4,5,5,6,7,7,8,9,NA)
cv(data, na.rm=TRUE)
```

datasource *Are values in memory and/or on disk?*

Description

These are helper functons for programmers and for debugging that provide information about whether a Raster object has associated values, and if these are in memory or on disk.

`fromDisk` is `TRUE` if the data source is a file on disk; and `FALSE` if the object only exists in memory.

`inMemory` is `TRUE` if all values are currently in memory (RAM); and `FALSE` if not (in which case they either are on disk, or there are no values).

`hasValues` is `TRUE` if the object has cell values.

Usage

```
fromDisk(x)
inMemory(x)
hasValues(x)
```

Arguments

x	Raster* object

Value

Logical value

Author(s)

Robert J. Hijmans

Examples

```
rs <- raster(system.file("external/test.grd", package="raster"))
inMemory(rs)
fromDisk(rs)
rs <- readAll(rs)
inMemory(rs)
fromDisk(rs)
rs <- rs + 1
inMemory(rs)
fromDisk(rs)
rs <- raster(rs)
inMemory(rs)
fromDisk(rs)
rs <- setValues(rs, 1:ncell(rs))
inMemory(rs)
fromDisk(rs)
rs <- writeRaster(rs, filename='test', overwrite=TRUE)
```

```
inMemory(rs)
fromDisk(rs)
```

`dataType` *Data type*

Description

Get the datatype of a RasterLayer object. The datatype determines the interpretation of values written to disk. Changing the datatype of a Raster* object does not directly affect the way they are stored in memory; but it does affect how values are read from file (unless values are read via rgdal). If you change the datatype of a RasterLayer and then read values from disk these may be completely wrong, so only do this for debugging or when the information in the header file was wrong. To set the datatype of a new file, you can give a 'datatype' argument to the functions that write values to disk (e.g. writeRaster).

Usage

```
dataType(x)
dataType(x) <- value
```

Arguments

`x`	A `RasterLayer` object
`value`	A data type (see below)

Details

Setting the data type is useful if you want to write values to disk. In other cases use functions such as round()

Datatypes are described by 5 characters. The first three indicate whether the values are integers, decimal number or logical values. The fourth character indicates the number of bytes used to save the values on disk, and the last character indicates whether the numbers are signed (i.e. can be negative and positive values) or not (only zero and positive values allowed)

The following datatypes are available:

Datatype definition	minimum possible value	maximum possible value
LOG1S	FALSE (0)	TRUE (1)
INT1S	-127	127
INT1U	0	255
INT2S	-32,767	32,767
INT2U	0	65,534
INT4S	-2,147,483,647	2,147,483,647
INT4U	0	4,294,967,296
FLT4S	-3.4e+38	3.4e+38
FLT8S	-1.7e+308	1.7e+308

For all integer types, except the single byte types, the lowest (signed) or highest (unsigned) value is used to store NA. Single byte files do not have NA values. Logical values are stored as signed single byte integers, they do have an NA value (-127)

INT4U is available but they are best avoided as R does not support 32-bit unsigned integers.

Value

a Raster* object

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
dataType(r)
s <- writeRaster(r, 'new.grd', datatype='INT2U', overwrite=TRUE)
dataType(s)
```

---

density *Density plot*

Description

Create a density plots of values in a RasterLayer

Usage

```
density(x, ...)
```

Arguments

x	RasterLayer object
...	Additional arguments passed to plot

Value

A density plot (and a density object, returned invisibly of plot=TRUE).

Methods

A full call as implemented here:
`density(x, layer, maxpixels=100000, plot=TRUE, main, ...)`
`layer` can be used to subset the layers to plot in a multilayer object (RasterBrick or RasterStack)
`maxpixels` is the maximum number of (randomly sampled) cells to be used for creating the density plot
see `plot` for the other arguments

Author(s)

Robert J. Hijmans

`dim` *Dimensions of a Raster* object*

Description

Get (or set) the number of rows, columns, and layers of a Raster* object. You cannot use this function to set the dimentions of a RasterStack object.

Value

Integer

Methods

`dim(x)`

If `x` is a Raster* object

`dim(x) <- value`

`x`	RasterLayer or RasterBrick)
`value`	row number, or row _and_ column number (for a RasterLayer and a RasterBrick); or a row and column number _a

Author(s)

Robert J. Hijmans

See Also

`ncell`, `extent`, `res`

Examples

```
r <- raster()
dim(r)
dim(r) <- c(18)
dim(r)
dim(r) <- c(18, 36)
dim(r)
b <- brick(r)
dim(b)
dim(b) <- c(10, 10, 5)
dim(b)
```

direction *Direction*

Description

The direction (azimuth) to or from the nearest cell that is not NA. The direction unit is in radians, unless you use argument degrees=TRUE.

Usage

```
direction(x, ...)
```

Arguments

x	a RasterLayer object
...	additional arguments. See Details.

Details

The following additional arguments can be passed, to replace default values for this function

filename	Filename for the output RasterLayer
from	Logical. Default is FALSE. If TRUE, the direction from (instead of to) the nearest cell that is not NA is returned
degrees	Logical. If FALSE (the default) the unit of direction is radians.
format	Character. Output file type. See writeRaster
datatype	Character. Output data type. See dataType
overwrite	Logical. If TRUE, "filename" will be overwritten if it exists
progress	Character. "text", "window", or "" (the default, no progress bar)

Value

A RasterLayer object

Author(s)

Robert J. Hijmans

See Also

distance, gridDistance

For the direction between points, see the azimuth function in the geosphere package

Examples

```
r <- raster(ncol=36,nrow=18)
r[] <- NA
r[306] <- 1
```

```
b <- direction(r)
#plot(b)
```

`disaggregate` *Disaggregate*

Description

Disaggregate a RasterLayer to create a new RasterLayer with a higher resolution (smaller cells). The values in the new RasterLayer are the same as in the larger original cells unless you specify `method="bilinear"`, in which case values are locally interpolated (using the `resample` function).

Usage

```
## S4 method for signature 'Raster'
disaggregate(x, fact=NULL, method='', filename='', ...)
```

Arguments

`x`	a Raster object
`fact`	integer. amount of disaggregation expressed as number of cells (horizontally and vertically). This can be a single integer or two integers c(x,y), in which case the first one is the horizontal disaggregation factor and y the vertical disaggreation factor. If a single integer values is supplied, cells are disaggregated with the same factor in x and y direction
`method`	Character. '' or 'bilinear'. If 'bilinear', values are locally interpolated (using the `resample` function
`filename`	Character. Output filename (optional)
`...`	Additional arguments as for `writeRaster`

Value

Raster object

Author(s)

Robert J. Hijmans

See Also

`aggregate`

Examples

```
r <- raster(ncols=10, nrows=10)
rd <- disaggregate(r, fact=c(10, 2))
ncol(rd)
nrow(rd)
r[] <- 1:ncell(r)
rd <- disaggregate(r, fact=c(4, 2), method='bilinear')
```

---

distance *Distance*

Description

Calculate the distance, for all cells that are NA, to the nearest cell that is not NA.

The distance unit is in meters if the RasterLayer is not projected (+proj=longlat) and in map units (typically also meters) when it is projected.

Usage

```
## S4 method for signature 'RasterLayer'
distance(x, filename='', doEdge=FALSE, ...)
```

Arguments

x	RasterLayer object
filename	Character. Filename for the output RasterLayer (optional)
doEdge	Logical. If TRUE, the edge function is called first. This may be efficient in cases where you compute the distance to large blobs. Calling edge determines the edge cells that matter for distance computation
...	Additional arguments as for writeRaster

Value

RasterLayer

Author(s)

Robert J. Hijmans

See Also

distanceFromPoints, gridDistance, pointDistance

See the gdistance package for more advanced distances, and the geosphere package for great-circle distances (and more) between points in longitude/latitude coordinates.

Examples

```
r <- raster(ncol=36,nrow=18)
r[] <- NA
r[500] <- 1
dist <- distance(r)
#plot(dist / 1000)
```

`distanceFromPoints` *Distance from points*

Description

The function calculates the distance from a set of points to all cells of a RasterLayer.

The distance unit is in meters if the RasterLayer is not projected (`+proj=longlat`) and in map units (typically meters) when it is projected.

Usage

```
distanceFromPoints(object, xy, filename='', ...)
```

Arguments

`object`	RasterLayer object
`xy`	Matrix of x and y coordinates, or a SpatialPoints* object.
`filename`	Filename for the output RasterLayer
`...`	Additional arguments as for `writeRaster`

Value

RasterLayer object

Author(s)

Robert J. Hijmans

See Also

`distance`, `gridDistance`, `pointDistance`

Examples

```
r <- raster(ncol=36,nrow=18)
xy = c(0,0)
dist <- distanceFromPoints(r, xy)
#plot(dist)
```

draw *Draw a line or polygon*

Description

Draw a line or polygon on a plot (map) and save it for later use. After calling the function, start clicking on the map. To finish, right-click and select 'stop'.

Usage

```
drawPoly(sp=TRUE, col='red', lwd=2, ...)
drawLine(sp=TRUE, col='red', lwd=2, ...)
```

Arguments

sp	logical. If TRUE, the output will be a sp object (SpatialPolygons or SpatialLines). Otherwise a matrix of coordinates is returned
col	the color of the lines to be drawn
lwd	the width of the lines to be drawn
...	additional arguments padded to locator

Value

If sp==TRUE a SpatialPolygons or SpatialLines object; otherwise a matrix of coordinates

Author(s)

Robert J. Hijmans

See Also

locator

drawExtent *Create an Extent object by drawing on a map*

Description

Click on two points of a plot (map) to obtain an object of class Extent ('bounding box')

Usage

```
drawExtent(show=TRUE, col="red")
```

Arguments

show	Logical. If TRUE, the extent will be drawn on the map
col	Sets the color of the lines of the extent

Value

an object of class Extent

Author(s)

Robert J. Hijmans

Examples

```
## Not run:
r1 <- raster(nrow=10, ncol=10)
r1[] <- runif(ncell(r1))
plot(r1)
e <- drawExtent()
# now click on the map twice
mean(values(crop(r1, drawExtent())))
# now click on the map twice

## End(Not run)
```

---

edge *Edge detection*

Description

Detect edges. Edges are cells that have more than one class in the 4 or 8 cells surrounding it, or, if classes=FALSE, cells with values and cells with NA.

Usage

```
## S4 method for signature 'RasterLayer'
edge(x, filename="", type='inner', classes=FALSE, directions=8, ...)
```

Arguments

x	RasterLayer object
filename	Character. Filename for the output RasterLayer (optional)
type	Character. 'inner', or 'outer'
classes	Character. Logical. If TRUE all different values are (after rounding) distinguished, as well as NA. If FALSE (the default) only edges between NA and non-NA cells are considered

directions	Integer. Which cells are considered adjacent? Should be 8 (Queen's case) or 4 (Rook's case)
...	Additional arguments as for writeRaster

Value

RasterLayer. Cell values are either 1 (and edge) or 0 (not an edge), or NA

Author(s)

Robert J. Hijmans

See Also

focal, clump

Examples

```
r <- raster(nrow=18, ncol=36, xmn=0)
r[150:250] <- 1
r[251:450] <- 2
plot( edge(r, type='inner') )
plot( edge(r, type='outer') )
plot( edge(r, classes=TRUE) )
```

---

expand *Expand*

Description

Expand returns an Raster* object with a larger spatial extent. The output Raster object has the outer minimum and maximum coordinates of the input Raster and Extent arguments. Thus, all of the cells of the original raster are included. See crop if you (also) want to remove rows or columns.

There is also an expand method for extent objects to enlarge (or reduce) an Extent. You can also use algebraic notation to do that (see examples)

Usage

```
## S4 method for signature 'Raster'
expand(x, y, value=NA, filename='', ...)

## S4 method for signature 'Extent'
expand(x, y, ...)
```

Arguments

x	Raster or Extent object
y	If x is a Raster object, y should be an Extent object, or any object that is or has an Extent object, or an object from which it can be extracted (such as sp objects). Alternatively, you can provide a vector of lenght 2 with the number indicating the amount of rows and columns that need to be added (or a single number when the number of rows and columns is equal) If x is an Extent object, y should be a numeric vector of 1, 2, or 4 elements
value	value to assign to new cells
filename	Character (optional)
...	Additional arguments as for writeRaster

Value

RasterLayer or RasterBrick, or Extent

Author(s)

Robert J. Hijmans and Etienne B. Racine (Extent method)

See Also

crop, merge

Examples

```
r <- raster(xmn=-150, xmx=-120, ymx=60, ymn=30, ncol=36, nrow=18)
r[] <- 1:ncell(r)
e <- extent(-180, 0, 0, 90)
re <- expand(r, e)

# expand with a number of rows and columns (at each side)
re2 <- expand(r, c(2,10))

# Extent object
e <- extent(r)
e
expand(e, 10)
expand(e, 10, -10, 0, 20)
e + 10
e * 2
```

`extension` *Filename extensions*

Description

Get or change a filename extension

Usage

```
extension(filename, value=NULL, maxchar=10)
extension(filename) <- value
```

Arguments

`filename`	A filename, with or without the path
`value`	A file extension with or without a dot, e.g., ".txt" or "txt"
`maxchar`	Maximum number of characters after the last dot in the filename, for that string to be considered a filename extension

Value

A file extension, filename or path.

If `ext(filename)` is used without a `value` argument, it returns the file extension; otherwise it returns the filename (with new extions set to `value`

Author(s)

Robert J. Hijmans

Examples

```
fn <- "c:/temp folder/filename.exten sion"
extension(fn)
extension(fn) <- ".txt"
extension(fn)
fn <- extension(fn, '.document')
extension(fn)
extension(fn, maxchar=4)
```

extent *Extent*

Description

This function returns an Extent object of a Raster* or Spatial* object (or an Extent object), or creates an Extent object from a matrix (2x2; rows=xmin, xmax; cols=ymin, ymax), vector (length=4; order= xmin, xmax, ymin, ymax) or list (with at least two elements, with names 'x' and 'y')

bbox returns a sp package like 'bbox' object (a matrix)

Usage

```
extent(x, ...)
```

Arguments

x	A Raster* or Extent object, a matrix, or a vector of four numbers
...	Additional arguments. When x is a single number you can pass three additional numbers (xmin, xmax, ymin, ymax) When x is a Raster* object, you can pass four additional arguments to crop the extent: r1, r2, c1, c2, representing the first and last row and column number

Value

An Extent object

Author(s)

Robert J. Hijmans; Etienne Racine write the extent function for a list

See Also

extent, drawExtent

Examples

```
r <- raster()
extent(r)
extent(c(0, 20, 0, 20))
#is equivalent to
extent(0, 20, 0, 20)
extent(matrix(c(0, 0, 20, 20), nrow=2))
x <- list(x=c(0,1,2), y=c(-3,5))
extent(x)

#crop the extent by row and column numbers
extent(r, 1, 20, 10, 30)
```

`Extent coordinates` *Coordinates of the Extent of a Raster object*

Description

These functions return or set the extreme coordinates of a Raster* object.

Usage

```
xmin(x)
xmax(x)
ymin(x)
ymax(x)

xmin(x) <- value
xmax(x) <- value
ymin(x) <- value
ymax(x) <- value
```

Arguments

`x`	A Raster* object
`value`	A new x or y coordinate

Value

a single number

Author(s)

Robert J. Hijmans

See Also

`extent`, `dimensions`

Examples

```
r <- raster(xmn=-0.5, xmx = 9.5, ncols=10)
xmin(r)
xmax(r)
ymin(r)
ymax(r)
xmin(r) <- -180
xmax(r) <- 180
```

`Extent math` *round Extent coordinates*

Description

use `round(x, digits=0)` to round the coordinates of an Extent object to the number of digits specified. This can be useful when dealing with a small inprecision in the data (e.g. 179.9999 instead of 180). `floor` and `ceiling` move the coordiantes to the outer or inner whole integer numbers.

It is also possible to use Arithmetic functions with Extent objects (but these work perhaps unexpectedly!)

See `Math-methods` for these (and many more) methods with Raster* objects.

Usage

```
## S4 method for signature 'Extent'
floor(x)
## S4 method for signature 'Extent'
ceiling(x)
```

Arguments

`x`	Extent object

Author(s)

Robert J. Hijmans

See Also

`Math-methods`

Examples

```
e <- extent(c(0.999999,  10.000011, -60.4, 60))
round(e)
ceiling(e)
floor(e)
```

Extent-class *Class "Extent"*

Description

Objects of class Extent are used to define the spatial extent (extremes) of objects of the BasicRaster and Raster* classes.

Objects from the Class

You can use the extent function to create Extent objects, or to extract them from Raster* and Spatial* objects.

Slots

xmin: minimum x coordinate

xmax: maximum x coordinate

ymin: minumum y coordinate

ymax: maximum y coordinate

Methods

show display values of a Extent object

Author(s)

Robert J. Hijmans

See Also

extent, setExtent

Examples

```
ext <- extent(-180,180,-90,90)
ext
```

`extract` *Extract values from Raster objects*

Description

Extract data from a Raster object using cell numbers or the locations of other spatial data (i.e., a spatial query). You can use coordinates (points), lines, polygons or an Extent (rectangle) object.

Usage

```
extract(x, y, ...)
```

Arguments

`x`	Raster* object
`y`	A vector (representing cell numbers); or a SpatialPoints object or a two-column data.frame or matrix (representing points); or a `SpatialPolygons`, `SpatialLines`, or `Extent` object
`...`	Additional arguments, see under Details

Details

Below are additional arguments that can be used, depending on the type of objects used.

= all objects =

1) `fun`. Function to summarize the values (e.g. `mean`). The function should take a single numeric vector as argument and return a single values (e.g. mean, min or max), and accept a `na.rm` argument. Thus, standard R functions not including an na.rm argument must be wrapped as in this example: fun=function(x,...)length(x)

If `y` represents points supplying a `fun` argument is only useful when a buffer is used (see below).

2) `na.rm`. Only useful when `fun` is supplied. If `na.rm=TRUE` (the default value), NA values are removed before fun is applied. This argument may be ignored if the function used has a `...` argument and ignres an additional `na.rm` argument.

= points =

If `y` represents points, `extract` returns the values of a Raster* object for the cells in which a set of points fall. Additional arguments that can be suplied to this function if `y` represents points:

1) `method`. If `method='simple'` (the default), values for the cell a point falls in are returned. The alternative is `method='bilinear'`, in which case the returned values are interpolated from the values of the four nearest raster cells.

2) `buffer`. The radius of a buffer around each point for which to extract cell values. If the distance between the sampling point and the center of a cell is less than or equal to the buffer, the cell is included. The buffer can be specified as a single value, or as a vector of the length of the number of points. If the data are not projected (latitude/longitude), the unit should be meters. Otherwise it should be in map-units (typically also meters).

3) cellnumbers. Logical. If cellnumbers=TRUE and buffer is not NULL, cellnumbers will also be returned. cellnumbers is set to FALSE when an argument fun is used.

= lines =

df. Logical. If df=TRUE, results will be returned as a data.frame, rather than as a list.

And, if no fun argument is supplied:

cellnumbers. Logical. If cellnumbers=TRUE is used, cell-numbers will also be returned.

= polygons =

If y represents polygons, the extract method returns the values of the cells of a Raster* object that is covered by a polygon. A cell is covered if its center is inside the polygon (but see the weights option for considering partly covered cells; and argument small for getting values for small polygons anyway).

if y represents polygons are:

1) small. Logical. Default is FALSE. If TRUE a value is also returned for relatively small polygons (e.g. those smaller than a single cell of the Raster* object), or polygons with an odd shape, for which otherwise no values are returned because they do not cover any Raster* object's cell centers. In some cases, you could use alternatively use the centroids of such polygons, for example using extract(x, coordinates(y)) or extract(x, coordinates(y), method='bilinear').

2) df. Logical. If df=TRUE, results will be returned as a data.frame, rather than as a list.

Additional arguments that can be suplied to this function, if no fun argument is supplied:

3) weights. If TRUE, the function returns, for each polygon, a matrix with the cell values and the approximate fraction of each cell that is covered by the polygon(rounded to 1/100). The weights can be used for averaging; see examples. This option can be useful if the polygons are small relative to the cells size of the Raster* object.

4) cellnumbers. Logical. If cellnumbers=TRUE, cell-numbers will also be returned.

= focal values =

If y is missing, you can supply two arguments to get 'focal' values (i.e. the values in a neighborhood around each cell)

1) row. row number (this is a required argument).

2) ngb. The neighborhood to consider. Either a single integer or a vector of two integers. Default is 3. See focal

= all multi-layer objects =

If x is a RasterStack or RasterBrick object, extract accepts, in addition to the arguments listed above, these two additional arguments:

1)layer. Integer. First layer for which you want values

2) nl. Integer. Number of layers for which you want values

Value

A vector for RasterLayer objects, and a matrix for a RasterStack or RasterBrick object. A list if y is a SpatialPolygons* or SpatialLines* object or if a buffer argument is used (but not a fun argument). focal values are returned as a n column matrix. The first column has the column numbers, the remaining has the focal values for each layer.

Author(s)

Robert J. Hijmans

See Also

values

Examples

```
r <- raster(ncol=36, nrow=18)
r[] <- 1:ncell(r)

###############################
# extract values by cell number
###############################
extract(r, c(1:2, 10, 100))
s <- stack(r, sqrt(r), r/r)
extract(s, c(1, 10, 100), layer=2, n=2)

###############################
# extract values with points
###############################
xy <- cbind(-50, seq(-80, 80, by=20))
extract(r, xy)

sp <- SpatialPoints(xy)
extract(r, sp, method='bilinear')

# examples with a buffer
extract(r, xy[1:3,], buffer=1000000)
extract(r, xy[1:3,], buffer=1000000, fun=mean)

## illustrating the varying size of a buffer (expressed in meters) on a longitude/latitude raster
 z <- extract(r, xy, buffer=1000000)
 s <- raster(r)
 for (i in 1:length(z)) { s[z[[i]]] <- i }
## compare with raster that is not longitude/latitude
 projection(r) <- "+proj=utm +zone=17"
 xy[,1] <- 50
 z <- extract(r, xy, buffer=8)
 for (i in 1:length(z)) { s[z[[i]]] <- i }
 plot(s)
# library(maptools)
# data(wrld_simpl)
# plot(wrld_simpl, add=TRUE)

###############################
# extract values with lines
###############################

cds1 <- rbind(c(-50,0), c(0,60), c(40,5), c(15,-45), c(-10,-25))
```

```
cds2 <- rbind(c(80,20), c(140,60), c(160,0), c(140,-55))
lines <- SpatialLines(list(Lines(list(Line(cds1)), "1"), Lines(list(Line(cds2)), "2") ))

extract(r, lines)

################################
# extract values with polygons
################################
cds1 <- rbind(c(-180,-20), c(-160,5), c(-60, 0), c(-160,-60), c(-180,-20))
cds2 <- rbind(c(80,0), c(100,60), c(120,0), c(120,-55), c(80,0))
polys <- SpatialPolygons(list(Polygons(list(Polygon(cds1)), 1), Polygons(list(Polygon(cds2)), 2)))

#plot(r)
#plot(polys, add=TRUE)
v <- extract(r, polys)
v
# mean for each polygon
unlist(lapply(v, function(x) if (!is.null(x)) mean(x, na.rm=TRUE) else NA ))

# v <- extract(r, polys, cellnumbers=TRUE)

# weighted mean
# v <- extract(r, polys, weights=TRUE, fun=mean)
# equivalent to:
# v <- extract(r, polys, weights=TRUE)
# sapply(v, function(x) if (!is.null(x)) {sum(apply(x, 1, prod)) / sum(x[,2])} else NA  )



################################
# extract values with an extent
################################
e <- extent(150,170,-60,-40)
extract(r, e)
#plot(r)
#plot(e, add=T)


################################
# focal values
################################
f <- extract(r, row=5, ngb=3)
f[1:15, ]
```

Extract by index *Indexing to extract values of a Raster* object*

Description

These are shorthand methods that call other methods that should normally be used, such as `getValues`, `extract`, `crop`.

`object[i]` can be used to access values of a Raster* object, using cell numbers. You can also use row and column numbers as index, using `object[i,j]` or `object[i,]` or `object[,j]`. In addition you can supply an Extent, SpatialPolygons, SpatialLines or SpatialPoints object.

If `drop=TRUE` (the default) cell values are returned (a vector for a RasterLayer, a matrix for a RasterStack or RasterBrick). If `drop=FALSE` a Raster* object is returned that has the extent covering the requested cells, and with all other non-requested cells within this extent set to `NA`.

If you supply a RasterLayer, its values will be used as logical (TRUE/FALSE) indices if both Raster objects have the same extent and resolution; otherwise the cell values within the extent of the RasterLayer are returned.

Double brackes '[[ ]]' can be used to extract one or more layers from a multi-layer object.

Methods

`x[i]`

`x[i,j]`

Arguments

`x`	a Raster* object
`i`	cell number(s), row number(s), a (logical) RasterLayer, Spatial* object
`j`	column number(s) (only available if i is (are) a row number(s))
`drop`	If `TRUE`, cell values are returned. Otherwise, a Raster* object is returned

See Also

`getValues`, `setValues`, `extract`, `crop`, `rasterize`

Examples

```
r <- raster(ncol=10, nrow=5)
r[] <- 1:ncell(r)

r[1]
r[1:10]
r[1,]
r[,1]
r[1:2, 1:2]

s <- stack(r, sqrt(r))
s[1:3]
s[[2]]
```

`extremeValues` *Minimum and maximum values*

Description

Returns the minimum or maximum value of a RasterLayer or layer in a RasterStack

Usage

```
minValue(x, ...)
maxValue(x, ...)
```

Arguments

`x`	RasterLayer or RasterStack object
`...`	Additional argument: layer number (for RasterStack or RasterBrick objects)

Details

If a Raster* object is created from a file on disk, the min and max values are often not known (depending on the file format). You can use `setMinMax` to set them in the Raster* object.

Value

a number

Author(s)

Robert J. Hijmans

Examples

```
r <- raster()
r <- setValues(r, 1:ncell(r))
minValue(r)
maxValue(r)
r <- setValues(r, round(100 * runif(ncell(r)) + 0.5))
minValue(r)
maxValue(r)

r <- raster(system.file("external/test.grd", package="raster"))
# this raster is from an 'external' format, so min and maxValue are not known
minValue(r)
maxValue(r)

r1 <- setMinMax(r)
# now they are known:
minValue(r1)
maxValue(r1)
```

```
r2 <- readAll(r)
# now they are known too:
minValue(r2)
maxValue(r2)
```

`factors` *Factors*

Description

Factors are categorical variables. These functions allow for defining a layer as a categorical variable. This feature is under development, and whether a layer is defined as a factor or not is currently ignored by all other functions.

Usage

```
is.factor(x)
asFactor(x, ...)
labels(object, ...)
labels(object) <- value
```

Arguments

`x`	A Raster* object
`object`	A Raster* object
`value`	list
`...`	Additional arguments

Value

Raster* object

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(nrow=10, ncol=10)
r[] <- (runif(ncell(r)) * 10)
is.factor(r)
r <- round(r)
f <- asFactor(r)
is.factor(f)
```

filename *Filename*

Description

Get the filename of a Raster* object. You cannot set the filename of an object (except for RasterStack objects); but you can provide a 'filename= ' argument to a function that creates a new RasterLayer or RasterBrick* object.

Usage

```
filename(x)
```

Arguments

x	A Raster* object

Value

a Raster* object

Author(s)

Robert J. Hijmans

Examples

```
r <- raster( system.file("external/test.grd", package="raster") )
filename(r)
```

filledContour *Filled contour plot*

Description

Filled contour plot of a RasterLayer. This is a wrapper around `filled.contour` for RasterLayer objects.

Usage

```
filledContour(x, y=1, maxpixels=100000, ...)
```

Arguments

x	A Raster* object
y	Integer. The layer number of x (if x has multiple layers)
maxpixels	The maximum number of pixels
...	Any argument that can be passed to filled.contour (graphics package)

Author(s)

Robert J. Hijmans

See Also

filled.contour, persp, plot

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
filledContour(r)
```

---

flip *Flip*

Description

Flip the values of a Raster* object by inverting the order of the rows (direction=y) or the columns direction='x'.

Usage

```
flip(x, direction, ...)
```

Arguments

x	a Raster* object
direction	Character. 'y' or 'x'; or 1 (=x) or 2 (=y)
...	Additional arguments as for writeRaster

Value

RasterLayer or RasterBrick

Author(s)

Robert J. Hijmans

See Also

transpose: t, rotate

Examples

```
r <- raster(nrow=18, ncol=36)
m <- matrix(1:ncell(r), nrow=18)
r[] <- as.vector(t(m))
rx <- flip(r, direction='x')
r[] <- as.vector(m)
ry <- flip(r, direction='y')
```

---

focal *Focal values*

Description

Calculate focal ("moving window") values for the neighborhood of focal cells using a matrix of weights, perhaps in combination with a function.

Usage

```
## S4 method for signature 'RasterLayer'
focal(x, w=3, fun, filename='', na.rm=FALSE, pad=FALSE, padValue=NA, NAonly=FALSE, ...)
```

Arguments

x	RasterLayer
w	matrix of weights (the moving window), e.g. a 3 by 3 matrix; see Details. The matrix can also be expressed as the number of cells in a single direction or in two directions from the focal cell, in which case the weights are all set to 1. I.e. w=3 refers to a 3 by 3 matrix: 2 cells at each side of the focal cell, queen's case, 9 cells in total. This is equivalent to w=c(3,3). You can also specify a rectangular neighborhood, e.g. w=c(3,5); but the sides must be odd numbers. If you need even sides, you can add a column or row with weights of zero.
fun	function (optional). The function fun should take multiple numbers, and return a single number. For example mean, modal, min or max. It should also accept a na.rm argument (or ignore it, e.g. as one of the 'dots' arguments. For example, length will fail, but function(x, ...){na.omit(length(x))} works.
filename	character. Filename for a new raster (optional)
na.rm	logical. If TRUE, NA will be removed from focal computations. The result will only be NA if all focal cells are NA. Except for some special cases (weights of 1, functions like min, max, mean), using na.rm=TRUE is generally not a good idea in this function because it will unbalance the effect of the weights
pad	logical. If TRUE, additional 'virtual' rows and columns are padded to x such that there are no edge effects. This can be useful when a function needs to have access to the central cell of the filter

padValue	logical. The value of the cells of the padded rows and columns
NAonly	logical. If TRUE, only cell values that are NA are replaced with the computed focal values
...	additional arguments. See Details.

Details

focal uses a matrix of weights for the neighborhood of the focal cells. The default function is sum. It is computationally much more efficient to adjust the weights-matrix than to use another function through the fun argument. Thus while the following two statements are equivalent (if there are no NA values), the first one is faster than the second one:

```
a <- focal2(x, w=matrix(1/9, nc=3, nc=3))
```

```
b <- focal2(x, w=3, fun=mean)
```

There is, however, a difference if NA values are considered. One can use the na.rm=TRUE option which may make sense when using a function like mean. However, the results would be wrong when using a weights matrix.

Laplacian filter: `filter=matrix(c(0,1,0,1,-4,1,0,1,0), nrow=3)`

Sobel filter: `filter=matrix(c(1,2,1,0,0,0,-1,-2,-1) / 4, nrow=3)`

The following additional arguments can be passed, to replace default values for this function

overwrite	Logical. If TRUE, "filename" will be overwritten if it exists
format	Character. Output file type. See writeRaster
datatype	Character. Output data type. See dataType
progress	Character. "text", "window", or "" (the default, no progress bar)

Value

RasterLayer

Note

This function has replaced a previous version of focal, and functions focalNA and focalFilter)

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(ncols=36, nrows=18, xmn=0)
r[] <- runif(ncell(r))

# 3x3 mean filter
r3 <- focal(r, w=matrix(1/9,nrow=3,ncol=3))

# 5x5 mean filter
r5 <- focal(r, w=matrix(1/25,nrow=5,ncol=5))
```

```
# Gaussian filter for square cells
fgauss <- function(sigma, n=5) {
m <- matrix(nc=n, nr=n)
col <- rep(1:n, n)
row <- rep(1:n, each=n)
x <- col - ceiling(n/2)
y <- row - ceiling(n/2)
# according to http://en.wikipedia.org/wiki/Gaussian_filter
m[cbind(row, col)] <- 1/(2*pi*sigma^2) * exp(-(x^2+y^2)/(2*sigma^2))
# sum of weights should add up to 1
m / sum(m)
}

gf=fgauss(1.5)
rg <- focal(r, w=gf)

# The max value for the lower-rigth corner of a 3x3 matrix around a focal cell
f = matrix(c(0,0,0,0,1,1,0,1,1), nrow=3)
f
rm <- focal(r, w=f, fun=max)

# global lon/lat data: no 'edge effect' for the columns
xmin(r) <- -180
r3g <- focal(r, w=matrix(1/9,nrow=3,ncol=3))
```

`freq` *Frequency table*

Description

Frequency table of the values of a RasterLayer.

Usage

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

Arguments

`x`	A RasterLayer object
`...`	Additional arguments. See under Details

Value

A matrix

Methods

A full call as implemented here:

`freq(x, digits=0, progress, ...)`

the `digits` argument is used passed to `round`

with `progress`, a progres bar can be specified. Choose from "text", "window", or "" (the default, no progress bar)

Author(s)

Robert J. Hijmans

See Also

`count`, `crosstab` and `zonal`

Examples

```
r <- raster(nrow=18, ncol=36)
r[] <- runif(ncell(r))
r <- r * r * r * 10
freq(r)
```

`Gain and offset` *Gain and offset of values on file*

Description

These functions can be used to get or set the gain and offset parameters used to transform values when reading from a file (currently these are ignored when writing a file). The gain and offset parameters are applied to the raw values using the formula below:

`value <- value * gain + offset`

The default value for gain is 1 and for offset is 0. 'gain' is sometimes referred to as 'scale'.

Note that setting gain and/or offset only affects reading of values from a file, it does not change values that are already (or only) in memory.

Usage

```
gain(x)
gain(x) <- value
offs(x)
offs(x) <- value
```

Arguments

`x`	Raster* object
`value`	Single numeric value

Value

Raster* object or numeric value(s)

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
gain(r)
offs(r)
r[1505:1510]
gain(r) <- 10
offs(r) <- 5
r[1505:1510]
```

---

getData *Get geographic data*

Description

Get geographic data for anywhere in the world. Data are read from files that are first downloaded if necessary.

Usage

```
getData(name, download=TRUE, path='', ...)
```

Arguments

name	Data set name, currently supported are 'GADM', 'countries', 'SRTM', 'alt', and 'worldclim'. See Details for more info
download	Logical. If TRUE data will be downloaded if not locally available
path	Character. Path name indicating where to store the data. Default is the current working directory
...	Additional required (!) parameters. These are data set specific. See Details

Details

'alt' stands for altitude (elevation); the data were aggregated from SRTM 90 m resolution data between -60 and 60 latitude. 'GADM' is a database of global administrative boundaries. 'worldclim' is a database of global interpolated climate data. 'SRTM' refers to the hole-filled CGIAR-SRTM (90 m resolution). 'countries' has polygons for all countries at a higher resolution than the 'wrld_simpl' data in the maptools pacakge .

If name is 'alt' or 'GADM' you must provide a 'country=' argument. Countries are specified by their 3 letter ISO codes. Use getData('ISO3') to see these codes. In the case of GADM you must also provide the level of administrative subdivision (0=country, 1=first level subdivision). In the case of alt you can set 'mask' to FALSE. If it is TRUE values for neighbouring countries are set to NA. For example:

```
getData('GADM', country='FRA', level=1)
```

```
getData('alt', country='FRA', mask=TRUE)
```

If name is 'SRTM' you must provide 'lon' and 'lat' arguments (longitude and latitude). These should be single numbers somewhere within the SRTM tile that you want.

```
getData('SRTM', lon=5, lat=45)
```

If name='worldclim' you must also provide a variable name 'var=', and a resolution 'res='. Valid variables names are 'tmin', 'tmax', 'prec' and 'bio'. Valid resolutions are 0.5, 2.5, 5, and 10 (minutes of a degree). In the case of res=0.5, you must also provide a lon and lat argument for a tile; for the lower resolutions global data will be downloaded. In all cases there are 12 (monthly) files for each variable except for 'bio' which contains 19 files.

```
getData('worldclim', var='tmin', res=0.5, lon=5, lat=45)
```

```
getData('worldclim', var='bio', res=10)
```

Value

A spatial object (Raster* or Spatial*)

Author(s)

Robert J. Hijmans

References

http://www.worldclim.org

http://www.gadm.org

http://srtm.csi.cgiar.org/

http://diva-gis.org/gdata

---

getValues *Get raster cell values*

Description

getValues returns all values or a row of values for a Raster* object. It will take them from memory if available, else it will read them from disk. Function values is a shorthand version of getValues.

Usage

```
getValues(x, row, nrows, ...)
values(x, ...)
```

Arguments

x	Raster* object
row	Numeric. Row number, should be between 1 and nrow(x), or missing in which case all values are returned
nrows	Numeric. Number of rows. Should be an integer number > 0, or missing
...	Additional arguments, see Details

Details

Additional arguments when x is a RasterLayer: format Character. specifiy the output format. Either "" or "matrix". The default is "", in which case a vector is returned. The values returned for a RasterStack or RasterBrick are always a matrix, with the rows representing cells, and the columns representing layers.

Value

vector or matrix of raster values

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
getValues(r)
getValues(r, row=10)
```

---

getValuesBlock *Get a block of raster cell values*

Description

getValuesBlock returns values for a block (rectangular area) of values of a Raster* object. It will take them from memory if available, else it will read them from disk.

Usage

```
getValuesBlock(x, row, ...)
```

Arguments

x	Raster* object
row	Numeric. Row number, should be between 1 and nrow(x), or missing in which case all values are returned
...	Additional arguments, see Details

Details

Additional arguments:

nrows	Integer. How many rows? Default is 1
col	Integer. Start column. Default is 1
ncols	Integer. How many columns? Default is the number of colums left after the start column
lyrs	Integer (vector). Which layers? Default is all layers (1:nlayers(x))

Value

vector (x=RasterLayer) or matrix

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
b <- getValuesBlock(r, row=100, nrows=3, col=10, ncols=5)
b
b <- matrix(b, nrow=3, ncol=5, byrow=TRUE)
b

logo <- brick(system.file("external/rlogo.grd", package="raster"))
getValuesBlock(logo, row=35, nrows=3, col=50, ncols=3, lyrs=2:3)
```

---

gridDistance *Grid distance*

Description

The function calculates the distance to cells of a RasterLayer when the path has to follow the center of neighboring raster cells (currently only implemented as a 'queen' case).

The distance is in meters if the RasterLayer is not projected (+proj=longlat) and in map units (typically meters) when it is projected.

Distances are calculated by summing local distances between cells, which are connected with their neighbours in 8 directions.

Usage

```
gridDistance(x, origin, omit=NULL, filename="", ...)
```

Arguments

x	a RasterLayer object
origin	value(s) of the cells from which the distance is calculated
omit	value(s) of the cells which cannot be traversed (optional)
filename	character. output filename (optional)
...	Additional arguments as for writeRaster

Details

If the RasterLayer to be processed is big, it will be processed in chunks automatically. This may lead to errors in the case of complex objects spread over different chunks (meandering rivers, for instance). You can try to solve these issues by varying the chunk size, see function setOptions().

Value

A RasterLayer object

Author(s)

Jacob van Etten and Robert J. Hijmans

See Also

See distance for 'as the crow flies' distance. Additional distance measures and options (directions, cost-distance) are available in the 'gdistance' package.

Examples

```
#world lonlat raster
r <- raster(ncol=10,nrow=10)
r[] <- 1
r[48] <- 2
r[66:68] <- 3
d <- gridDistance(r,origin=2,omit=3)
plot(d)

#UTM small area
projection(r) <- "+proj=utm +zone=15 +ellps=GRS80 +datum=NAD83 +units=m +no_defs"
d <- gridDistance(r,origin=2,omit=3)
plot(d)
```

hdr *Header files*

Description

Write header files to use together with raster binary files to read the data in other applications.

Usage

```
hdr(x, format, extension='.wld')
```

Arguments

x	RasterLayer or RasterBrick object associated with a binary values file on disk
format	Type of header file: 'VRT', 'BIL', 'ENVI', 'ErdasRaw', 'IDRISI', 'SAGA', 'RASTER', 'WORLDFILE', 'PRJ'
extension	File extension, only used with an ESRI worldfile (`format='WORLDFILE'`)

Details

The RasterLayer object must be associated with a file on disk.

You can use `writeRaster` to save a existing file in another format. But if you have a file in a 'raster' format (or similar), you can also only export a header file, and use the data file (.gri) that already exists. writeHeader can write a VRT (GDAL virtual raster) header (.vrt); an ENVI or BIL header (.hdr) file; an Erdas Raw (.raw) header file; an IDRISI (.rdc) or SAGA (.sgrd). This (hopefully) allows for reading the binary data (.gri), perhaps after changing the file extension, in other programs such as ENVI or ArcGIS.

Author(s)

Robert J. Hijmans

See Also

`writeRaster`, `writeGDAL`

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
r <- writeRaster(r, filename='export.grd', overwrite=TRUE)
hdr(r, format="ENVI")
```

head *Show the head or tail of a Raster* object*

Description

Show the head (first rows/columns) or tail (last rows/columns) of the cell values of a Raster* object.

Usage

```
head(x, ...)
tail(x, ...)
```

Arguments

x	Raster* object
...	Additional arguments: `rows=10` and `cols=20`, to set the maximum number of rows and columns that is shown. For RasterStack and RasterBrick objects there is an additional argument `lyrs`

Value

matrix

Author(s)

Robert J. Hijmans

See Also

`getValuesBlock`

Examples

```
r <- raster(nrow=25, ncol=25)
r[] = 1:ncell(r)
head(r)
tail(r, cols=10, rows=5)
```

hillShade *Hill shading*

Description

Compute hill shade from slope and aspect layers (both in radians). Slope and aspect can be computed with function terrain.

A hill shade layer is often used as a backdrop on top of which another, semi-transparant, layer is drawn.

Usage

```
hillShade(slope, aspect, angle=45, direction=0, filename='', normalize=FALSE, ...)
```

Arguments

slope	RasterLayer object with slope values (in radians)
aspect	RasterLayer object with aspect values (in radians)
angle	The the elevation angle of the light source (sun), in degrees
direction	The direction (azimuth) angle of the light source (sun), in degrees
filename	Character. Optional filename
normalize	Logical. If TRUE, values below zero are set to zero and the results are multiplied with 255
...	Standard additional arguments for writing RasterLayer files

Author(s)

Andrew Bevan, Robert J. Hijmans

References

Horn, B.K.P., 1981. Hill shading and the reflectance map. Proceedings of the IEEE 69(1):14-47

See Also

terrain

Examples

```
## Not run:
alt <- getData('alt', country='CHE')
slope <- terrain(alt, opt='slope')
aspect <- terrain(alt, opt='aspect')
hill <- hillShade(slope, aspect, 40, 270)
plot(hill, col=grey(0:100/100), legend=FALSE, main='Switzerland')
plot(alt, col=rainbow(25, alpha=0.35), add=TRUE)

## End(Not run)
```

hist *Histogram*

Description

Create a histogram of the values of a RasterLayer. NA values are ignored. Large datasets are sampled using `maxpixels`.

Usage

```
hist(x, ...)
```

Arguments

`x`	A Raster* object
`...`	Additional arguments. See under Methods and at `hist`

Value

This function is principally used for the side-effect of plotting a histogram, but it also returns an S3 object of class 'histogram' (invisibly if `plot=TRUE`).

Methods

A full call as implemented here:

`hist(x, layer, maxpixels=100000, plot=TRUE, main, ...)`

`layer` can be used to subset the layers to plot in a multilayer object (RasterBrick or RasterStack)

`maxpixels` is the maximum number of (randomly sampled) cells to be used for creating the histogram

see `hist` for the other arguments

See Also

`pairs`, `boxplot`

Examples

```
r1 <- raster(nrows=50, ncols=50)
r1 <- setValues(r1, runif(ncell(r1)))
r2 <- setValues(r1, runif(ncell(r1)))
rs <- r1 + r2
rp <- r1 * r2
par(mfrow=c(2,2))
plot(rs, main='sum')
plot(rp, main='product')
hist(rs)
```

```
a = hist(rp)
a
```

---

image *Image*

Description

Create an "image" type plot of a RasterLayer. This is an implementation of a generic function in the graphics package. In most cases the plot function would be preferable because it produces a legend (and has some additional options).

Usage

```
image(x, ...)
```

Arguments

x	A Raster* object
...	Any argument that can be passed to image (graphics package)

Methods

```
image(x, y=1, z=NULL, ...)
```

x	a Raster* object
...	

Author(s)

Robert J. Hijmans

See Also

image, contour, plot

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
image(r)
```

`inifile` *Read a .ini file*

Description

This function reads `'.ini'` files. These are text file databases that are organized in [sections] containing pairs of "name = value".

Usage

```
readIniFile(filename, token='=', commenttoken=';', aslist=FALSE, case)
```

Arguments

`filename`	Character. Filename of the .ini file
`token`	Character. The character that seperates the "name" (variable name) from the "value"
`commenttoken`	Character. This token and everything that follows on the same line is considered a 'comment' that is not for machine consumption and is ignored in processing
`aslist`	Logical. Should the values be returned as a list
`case`	Optional. Function that operates on the text, such as `toupper` or `tolower`

Details

This function allows for using inistrings that have "=" as part of a value (but the token cannot be part of the 'name' of a variable!). Sections can be missing.

Value

A n*3 matrix of characters with columns: section, name, value; or a list if `aslist=TRUE`.

Author(s)

Robert J. Hijmans

initialize *Intitialize*

Description

Create a new RasterLayer with values reflecting a cell property: v from 'x', 'y', 'col', 'row', or 'cell'. Alternatively, a function can be used. In that case, cell values are initialized without reference to pre-existing values. E.g., initialize with a random number (fun=runif). Either supply an argument to fun, or to v, but not both.

Usage

```
init(x, fun, v, filename="", ...)
```

Arguments

x	A Raster* object
fun	The function to be applied. This must be a function that can take the number of cells as a single argument to return a vector of values with a length equal to the number of cells
v	'x', 'y', 'row', 'col', or 'cell'
filename	Output filename
...	Additional arguments as for writeRaster

Value

RasterLayer

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(ncols=36, nrows=18)

x <- init(r, v='cell')

y <- init(r, fun=runif)

# there are different ways to set all values to 1
# for large rasters:
set1f <- function(x){rep(1, x)}
z1 <- init(r, fun=set1f, filename='test.grd', overwrite=TRUE)

# equivalent to
z2 <- setValues(r, rep(1, ncell(r)))
# or
```

```
r[] <- rep(1, ncell(r))
# or
r[] <- 1
```

`interpolate` *Interpolate*

Description

Make a RasterLayer with interpolated values based on a a fitted model object of classes such as 'gstat' or 'Krige'. I.e. these are models that have 'x' and 'y' as independent variables. If x and y are the only independent variables provide an empty (no associated data in memory or on file) RasterLayer for which you want predictions. If there are more spatial predictor variables provide these as a Raster* object in the first argument of the function. If you do not have x and y locations as implicit predictors in your model you should use `predict` instead.

Usage

```
interpolate(object, ...)
```

Arguments

`object`	a Raster* object
`...`	Additional arguments. See below, under Methods

Value

RasterLayer object

Methods

A full call to the method is

```
interpolate(object, model, filename='', fun=predict, xyOnly=TRUE, ext=NULL, const=NULL,
index=1, na.rm=TRUE, ...)
```

`object`	Raster* object
`model`	Fitted model object
`filename`	Output filename for a new raster; if NA the result is not written to a file but returned with the RasterLayer obje
`fun`	Function. Default value is 'predict', but can be replaced with e.g. 'predict.se' (depending on the class of the m
`xyOnly`	Logical. If `TRUE`, values of the Raster* object are not considered as co-variables; and only x and y (longitude a
`ext`	An Extent object to limit the prediction to a sub-region of `x`
`const`	data.frame. Can be used to add a constant for which there is no Raster object for model predictions. Particulal
`index`	Integer. To select the column if 'predict.model' returns a matrix with multiple columns
`na.rm`	Logical. Remove cells with NA values in the predictors before solving the model (and return a NA value for th
`...`	Additional arguments as for writeRaster

Author(s)

Robert J. Hijmans

See Also

predict, predict.gstat, Tps

Examples

```
## Not run:

## Thin plate spline interpolation with x and y only
library(fields)
r <- raster(system.file("external/test.grd", package="raster"))
ra <- aggregate(r, 10)
xy <- data.frame(xyFromCell(ra, 1:ncell(ra)))
v <- getValues(ra)
tps <- Tps(xy, v)
p <- raster(r)
p <- interpolate(p, tps)
p <- mask(p, r)
plot(p)
se <- interpolate(p, tps, fun=predict.se)
se <- mask(se, r)
plot(se)


##gstat examples
library(gstat)
## inverse distance weighted interpolation with gstat
r <- raster(system.file("external/test.grd", package="raster"))
data(meuse)
mg <- gstat(id = "zinc", formula = zinc~1, locations = ~x+y, data=meuse, nmax=7, set=list(idp = .5))
z <- interpolate(r, mg)
z <- mask(z, r)

## kriging
coordinates(meuse) = ~x+y
v <- variogram(log(zinc)~1, meuse)
m <- fit.variogram(v, vgm(1, "Sph", 300, 1))
g <- gstat(NULL, "log.zinc", log(zinc)~1, meuse, model = m)
projection(r) <- projection(meuse)
x <- interpolate(r, g)

## End(Not run)
```

intersect *Intersect Extent*

Description

Intersect two Extent objects. Returns the intersection, i.e. the area of overlap of two Extent objects. The second argument can also be any argument from which an Extent object can be extracted.

If the first object is a Raster* object, this function is equivalent to crop.

Usage

```
## S4 method for signature 'Extent'
intersect(x, y)

## S4 method for signature 'Raster'
intersect(x, y)
```

Arguments

x	Extent or Raster* object
y	Extent object, or any object from which an Extent can be extracted

Value

Extent or Raster* object

Author(s)

Robert J. Hijmans

See Also

union, extent, crop

Examples

```
e1 <- extent(-10, 10, -20, 20)
e2 <- extent(0, 20, -40, 5)
intersect(e1, e2)
```

intersectExtent *Extent intersection*

Description

Obsolete. See intersect

Usage

```
intersectExtent(x, ..., validate=TRUE)
```

Arguments

x	Extent object or object from which it can be coerced via extent (Raster* or Spatial* objects)
...	Additional Extent or Raster* or Spatial* objects
validate	Logical. If TRUE, an error is returned if the intersection is empty; else NULL is returned if the intersection is empty

Value

Extent object

Author(s)

Robert J. Hijmans

See Also

union, intersect, extent

Examples

```
e1 <- extent(-10, 10, -20, 20)
e2 <- extent(0, 20, -40, 5)
intersectExtent(e1, e2)
```

isLonLat — *Is this longitude/latitude data?*

Description

Test whether a Raster* object has the a longitude/latitude coordinate reference system.

Usage

```
isLonLat(x)
```

Arguments

x	Raster* object

Value

Logical

Author(s)

Robert J. Hijmans

Examples

```
r <- raster()
isLonLat(r)
projection(r) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"
isLonLat(r)
```

KML — *Write a KML or KMZ file*

Description

Export raster data to a KML file and an accompanying PNG image file. Multi-layer objects can be used to create an animation. The function attempts to combine these into a single (and hence more convenient) KMZ file (a zip file containing the KML and PNG files).

Usage

```
KML(x, ...)
```

Arguments

x	Raster* object
...	Additional arguments (see Details)

Details

You can use the following additional argumetns:

filename	output filename
time	character vector with time lables for multilayer objects. The lenght of this vector should be nlayers(x) to indic
col	color scheme to be used (see image
maxpixels	maximum number of pixels. If ncell(raster) > maxpixels, sampleRegular is used to reduce the number of pixel
zip	If there is no zip program on your path (on windows), you can supply the full path to a zip.exe here, in order t
...	Additional arguments that can be passed to image

Value

None. Used for the side-effect files written to disk.

Author(s)

This function was adapted for the raster package by Robert J. Hijmans, with ideas from Tony Fischbach, and based on functions in the maptools package by Duncan Golicher, David Forrest and Roger Bivand.

Examples

```
## Not run:
# Meuse data from the sp package
data(meuse.grid)
b <- rasterFromXYZ(meuse.grid)
projection(b) <- "+init=epsg:28992 +towgs84=565.237,50.0087,465.658,-0.406857,0.350733,-1.87035,4.0812
            +proj=sterea +lat_0=52.15616055555555 +lon_0=5.38763888888889 +k=0.9999079 +x_0=155000
  +y_0=463000 +ellps=bessel +units=m +no_defs"

# transform to longitude/latitude
p <- projectRaster(b, crs="+proj=longlat +datum=WGS84", method='ngb')
KML(p, file='meuse.kml')

## End(Not run)
```

Logic-methods *Logical operators and functions*

Description

The following logical (boolean) operators are available for computations with RasterLayer objects:

`&, |, and !`

The following functions are available with a Raster* argument:

`is.na`, `is.nan`, `is.finite`, `is.infinite`

Value

A Raster object with logical (TRUE/FALSE values)

Note

These are convenient operators/functions that are most usful for relatively small RasterLayers for which all the values can be held in memory. If the values of the output RasterLayer cannot be held in memory, they will be saved to a temporary file. In that case it could be more efficient to use calc instead.

Author(s)

Robert J. Hijmans

See Also

Math-methods, overlay, calc

Examples

```
r <- raster(ncols=10, nrows=10)
r[] <- runif(ncell(r)) * 10
r1 <- r < 3 | r > 6
r2 <- !r1
r3 <- r >= 3 & r <= 6
r4 <- r2 == r3
r[r>3] <- NA
r5 <- is.na(r)
r[1:5]
r1[1:5]
r2[1:5]
r3[1:5]
```

---

mask *Mask values in a Raster object*

Description

Create a new Raster* object where all cells that are NA in a 'mask' object are set to NA, and that has the same values as x in the other cells. Or use inverse=TRUE to set the cells that are not NA in the mask to NA.

Usage

```
mask(x, mask, ...)
```

Arguments

x	Raster* object
mask	Raster* object or a Spatial* object
...	Additional arguments. See under Methods

Value

A new Raster* object, and in some cases the side effect of a new file on disk.

Methods

The following additional arguments can be passed, to replace default values for this function

reverse	Boolean. If TRUE, areas on mask that are _not_ NA are masked.
filename	Character. output filename
format	Character. Output file type. See writeRaster
datatype	Character. Output data type. See dataType
overwrite	Logical. If TRUE, "filename" will be overwritten if it exists
progress	Character. "text", "window", or "" (the default, no progress bar)

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(ncol=10, nrow=10)
m <- raster(ncol=10, nrow=10)
r[] <- runif(ncell(r)) * 10
m[] <- runif(ncell(r))
m[m <0.5] <- NA
mr <- mask(r, m)
```

---

match *Value matching for Raster* objects*

Description

match returns a Raster* object with the position of the matched values. The cell values are the index of the table argument.

%in% returns a logical Raster* object indicating if the cells values were matched or not.

Usage

```
match(x, table, nomatch = NA_integer_, incomparables = NULL)

x %in% table
```

Arguments

x	Raster* object
table	vector of the values to be matched against
nomatch	the value to be returned in the case when no match is found. Note that it is coerced to integer
incomparables	a vector of values that cannot be matched. Any value in x matching a value in this vector is assigned the nomatch value. For historical reasons, FALSE is equivalent to NULL

Value

Raster* obeject

Author(s)

Robert J. Hijmans

See Also

calc, match

Examples

```
r <- raster(nrow=10, ncol=10)
r[] <- 1:100
m <- match(r, c(5:10, 50:55))
n <- r %in% c(5:10, 50:55)
```

Math-methods *Mathematical functions*

Description

Generic mathematical functions that can be used with a Raster* object as argument: "abs", "sign", "sqrt", "ceiling", "floor", "trunc", "cummax", "cummin", "cumprod", "cumsum", "log", "log10", "log2", "log1p", "acos", "acosh", "asin", "asinh", "atan", "atanh", "exp", "expm1", "cos", "cosh", "sin", "sinh", "tan", "tanh".

Note

You can use the, somewhat more flexible, function calc instead of the Math-methods.

Author(s)

Robert J. Hijmans

See Also

Arith-methods, calc, overlay, atan2

Examples

```
r1 <- raster(nrow=10, ncol=10)
r1 <- (setValues(r1, runif(ncell(r1)))) * 10
r2 <- sqrt(r1)
s <- stack(r1, r2) - 5
b <- abs(s)
```

---

merge *Merge*

Description

Merge Raster* objects to form a new Raster object with a larger spatial extent. If objects overlap, the values get priority in the same order as the arguments, but NA values are ignored (except when overlap=FALSE)

Usage

```
## S4 method for signature 'Raster,Raster'
merge(x, y, ..., tolerance=0.05, filename="", format, datatype, overwrite, progress, overlap=TRUE, ext=
```

Arguments

x	Raster* or Extent object
y	Raster* or Extent object
...	Additional Raster or Extent objects
tolerance	Numeric. Permissable difference in origin (relative to the cell resolution). See all.equal
filename	Character. output filename
overwrite	Logical. If TRUE, "filename" will be overwritten if it exists
format	Character. output file type. Either 'raster', 'ascii' or a supported GDAL 'driver' name see writeFormats
datatype	Character. Output data type. See dataType
progress	Character. "text", "window", or "" (the default, no progress bar)
overlap	Logical. If FALSE values of overlapping objects are based on the first layer, even if they are NA
ext	Extent object (optional) to limit the output to that extent

Details

The Raster objects must have the same origin and resolution. In areas where the Raster objects overlap, the values of the Raster object that is first in the sequence of arguments will be retained. If you'd rather use the average of cell values, or do another computation, you can use `mosaic` instead of merge.

Value

RasterLayer or RasterBrick

Author(s)

Robert J. Hijmans

Examples

```
r1 <- raster(xmx=-150, ymn=60, ncols=30, nrows=30)
r1[] <- 1:ncell(r1)
r2 <- raster(xmn=-100, xmx=-50, ymx=50, ymn=30)
res(r2) <- c(xres(r1), yres(r1))
r2[] <- 1:ncell(r2)
rm <- merge(r1, r2)
```

---

modal *modal value*

Description

Compute the mode for a vector of numbers, or across raster layers. The mode, or modal value, is the most frequent value in a set of values.

Usage

```
## S4 method for signature 'ANY'
modal(x, ..., ties='random', na.rm = FALSE)

## S4 method for signature 'Raster'
modal(x, ..., ties='random', na.rm = FALSE)
```

Arguments

x	A vector of numbers (typically integers for modal), or a Raster* object
...	additional (vectors of) numbers, or additional Raster* objects
ties	character. Indicates how to treat ties. Either 'random', 'lowest', 'highest', or 'NA'
na.rm	Remove (ignore) NA values

Value

vector or RasterLayer

Author(s)

Robert J. Hijmans

Examples

```
data <- c(0,1,2,3,3,3,3,4,4,4,5,5,6,7,7,8,9,NA)
modal(data, na.rm=TRUE)
```

`mosaic` *Merge Raster* objects, using a function to compute cell value for overlapping areas.*

Description

Mosaic Raster* objects to form a new object with a larger spatial extent. A functions is used to compute cell values in areas where layers overlap (in contrast to the `merge` function which uses the values of the 'upper' layer). All objects must have the same origin, resolution, and coordinate reference system.

Usage

```
mosaic(x, y, ...)
```

Arguments

`x`	Raster* object
`y`	Raster* object
`...`	Additional RasterLayers and other arguments. See below, under Methods

Details

The RasterLayer objects must have the same origin and resolution.

Value

RasterLayer or RasterBrick object.

Methods

A full call to mosaic is:

```
mosaic(x, y, ..., fun, na.rm=TRUE, tolerance=0.05, filename="", overwrite, format, progress )
```

Raster* object; or a list of Raster* objects

`y` Raster* object

`...` Additional Raster* objects

`fun` Function. E.g. mean, min, or max. Must be a function that accepts a 'na.rm' argument

`na.rm` Logical. Only return `NA` when all values are `NA`

`tolerance` Numeric. Permissable difference in origin (relative to the cell resolution). See all.equal

`filename` Character. output filename

`overwrite` Logical. If TRUE, "filename" will be overwritten if it exists

`format` Character. output file type. Either 'raster', 'ascii' or a supported GDAL 'driver' name see `writeFormats`

`datatype` Character. Output data type. See dataType

`progress` Character. "text", "window", or "" (the default, no progress bar)

Author(s)

Robert J. Hijmans

See Also

`merge`, `expand`

Examples

```
r <- raster(ncol=100, nrow=100)
r1 <- crop(r, extent(-10, 11, -10, 11))
r2 <- crop(r, extent(0, 20, 0, 20))
r3 <- crop(r, extent(9, 30, 9, 30))

r1[] <- 1:ncell(r1)
r2[] <- 1:ncell(r2)
r3[] <- 1:ncell(r3)

m1 <- mosaic(r1, r2, r3, fun=mean)

s1 <- stack(r1, r1*2)
s2 <- stack(r2, r2/2)
s3 <- stack(r3, r3*4)
m2 <- mosaic(s1, s2, s3, fun=min)
```

movingFun *Moving functions*

Description

Helper function to compute 'moving' functions, such as the 'moving average'

Usage

```
movingFun(x, n, fun=mean, type='around', circular=FALSE, na.rm=FALSE)
```

Arguments

x	A vector of numbers
n	Size of the 'window', i.e. the number of sequential elements to use in the function
fun	A function like mean, min, max, sum
type	Character. One of 'around', 'to', or 'from'. The choice indicates which values should be used in the computation. The focal element is always used. If type is 'around', the other elements are before and after the focal element. Alternatively, you can select the elements preceding the focal element ('to') or those coming after it 'from'. For example, to compute the movingFun with n=3 for element 5 of a vector; 'around' used elements 4,5,6; 'to' used elements 3,4,5, and 'from' uses elements 5,6,7
circular	Logical. If true, the data are considered to have a circular nature (e.g. months of the year), and the last elements in vector x are used in the computation of the moving function of the first element(s) of the vector, and the first elements are used in the computation of the moving function for the last element(s)
na.rm	Logical. If TRUE, NA values should be ingored (by fun)

Value

Numeric

Author(s)

Robert J. Hijmans (who was inspired by Diethelm Wuertz' rollFun function in the fTrading package)

Examples

```
movingFun(1:12, 3, mean)
movingFun(1:12, 3, mean, 'to')
movingFun(1:12, 3, mean, 'from')
movingFun(1:12, 3, mean, circular=TRUE)
```

```
v <- c(0,1,2,3,3,3,3,4,4,4,5,5,6,7,7,8,9,NA)
movingFun(v, n=5)
movingFun(v, n=5, na.rm=TRUE)
```

NAvalue *Set the NA value of a RasterLayer*

Description

NAvalue returns the value that is used to write NA values to disk (in 'raster' type files). If you set the NA value of a RasterLayer, this value will be interpreted as NA when reading the values from a file. Values already in memory will not be affected.

If the NA value is smaller than zero, all values smaller or equal to that number will be set to NA.

Usage

```
NAvalue(x) <- value
```

Arguments

x	A RasterLayer object
value	the value to be interpreted as NA; set this before reading the values from the file. Integer values are matched exactely; for decimal values files any value <= the value will be interpreted as NA

Value

Returns or set the NA value used for storage on disk.

Author(s)

Robert J. Hijmans

Examples

```
r1 <- raster(system.file("external/rlogo.grd", package="raster"))
r2 <- r1
NAvalue(r2)
NAvalue(r2) <- 255
#plot(r1)
#x11()
#plot(r2)
```

ncell *Number or rows, columns, and cells of a Raster* object*

Description

Get the number of rows, columns, or cells of a Raster* object.

Usage

```
ncol(x)
nrow(x)
ncell(x)
ncol(x) <- value
nrow(x) <- value
```

Arguments

x	a Raster object
value	row or column number (integer > 0)

Value

Integer

Author(s)

Robert J. Hijmans

See Also

dim, extent, res

Examples

```
r <- raster()
ncell(r)
ncol(r)
nrow(r)
dim(r)

nrow(r) <- 18
ncol(r) <- 36
# equivalent to
dim(r) <- c(18, 36)
```

nlayers *Number of layers*

Description

Get or set layer properties of a RasterStack object

Usage

```
nlayers(object)
layerNames(object)
layerNames(object) <- value
```

Arguments

object	Raster* object
value	a vector of character

Value

nlayers: A single numeric value >= 1 layerNames: a vector of character

Author(s)

Robert J. Hijmans

See Also

bands

Examples

```
#using a new default raster (1 degree global)
r <- raster(ncols=10, nrows=10)
r[] <- 1:ncell(r)
s <- stack(r, r, r)
nlayers(s)
layerNames(s)
layerNames(s) <- c('a', 'b', 'c')
layerNames(s)[2] <- c('bb')
```

obsolete *Obsolete functions*

Description

Obsolete functions. Use extract or rasterize instead.

Usage

```
polygonValues(...)
lineValues(...)
xyValues(...)
pointsToRaster(...)
linesToRaster(...)
polygonsToRaster(...)
```

Arguments

...	arguments

Value

An error message

See Also

extract, rasterize

Options *Global options for the raster package*

Description

setOptions allows you to set, and inspect, a number of global options used by the raster package.

Most of these options are used when writing files to disk. They can be ignored by specific functions if the corresponding argument is provided as an argument to these functions.

The default location is returned by rasterTmpDir. It is the same as that of the R tempdir but you can change it (for the current session) with rasterOptions(tmpdir= .

To permanentely set any of these options, you can add them to <your R installation>/etc/Rprofile.site. For example, to change the default directory used to save temporary files, add a line like this: options(rasterTmpDir='c:/temp/') to that file. All raster files in that folder that are older than 24 hrs are deleted when the raster package is loaded.

showOptions shows the current options.

saveOptions attempts to save the current options (with the exception of the 'todisk' option) to the /etc/Rprofile.site file to make them persistent (they will be set when starting a new R session).

clearOptions resets the options to their default values.

Usage

```
setOptions(format, overwrite, datatype, tmpdir, progress, timer, chunksize,
maxmemory, todisk, setfileext, tolerance)
showOptions()
saveOptions()
clearOptions()
```

Arguments

format	Character. The default file format to use. See writeFormats
overwrite	Logical. The default value for overwriting existing files. If TRUE, existing files will be overwritten
datatype	Character. The default data type to use. See dataType
tmpdir	Character. The default location for writing temporary files; See rasterTmpFile
progress	Character. Valid values are "text", "window" and "" (the default in most functions, no progress bar)
timer	Logical. If TRUE, the time it took to complete the function is printed
chunksize	Integer. Maximum number of cells to read/write in a single chunk while processing (chunk by chunk) disk based Raster* objects
maxmemory	Integer. Maximum number of cells to read into memory. I.e., if a Raster* object has more than this number of cells, canProcessInMemory will return FALSE
todisk	Logical. For debugging only. Default is FALSE and should normally not be changed. If TRUE, results are always written to disk, even if no filename is supplied (a temporary filename is used)
setfileext	Logical. Default is TRUE. If TRUE, the file extension will be changed when writing (if known for the file type). E.g. GTiff files will be saved with the .tif extension
tolerance	Numeric. The tolerance used when comparing the origin and resolution of Raster* objects. Expressed as the fraction of a single cell. This should be a number between 0 and 0.5

Value

None. This function is used for the side-effect of setting global options.

Author(s)

Robert J. Hijmans

See Also

options, rasterTmpFile

origin *Origin*

Description

Origin returns the coordinates of the point of origin of a Raster* object. This is the point closest to (0, 0) that you could get if you moved towards that point in steps of the x and y resolution.

Usage

```
origin(x)
```

Arguments

x	Raster* object

Value

A vector of two numbers (x and y coordinates).

Author(s)

Robert J. Hijmans

See Also

ncell, coordinates

Examples

```
r <- raster(xmn=-0.5, xmx = 9.5, ncols=10)
origin(r)
```

overlay *Overlay Raster objects*

Description

Create a new Raster* object, based on two or more Raster* objects. (You can also use a single object, but perhaps calc is what you are looking for in that case).

You should supply a function fun to set the way that the RasterLayers are combined. The number of arguments in the function must match the number of Raster objects (or take any number). For example, if you combine two RasterLayers you could use multiply: `fun=function(x,y){return(x*y)}` percentage: `fun=function(x,y){return(100 * x / y)}`. If you combine three layers you could use `fun=function(x,y,z){return((x + y) * z)}`

Note that the function must work for vectors (not only for single numbers). That is, it must return the same number of elements as its input vectors. Alternatively, you can also supply a function such as sum, that takes n arguments (as `'...'`), and perhaps also has a `na.rm` argument, like in `sum(..., na.rm)`.

If a single mutli-layer object is provided, its layers are treated as individual RasterLayer objects if the argument "unstack=TRUE" is used. If multiple objects are provided, they should have the same number of layers, or it should be possible to recycle them (e.g., 1, 3, and 9 layers, which would return a RasterBrick with 9 layers).

Usage

```
## S4 method for signature 'Raster,Raster'
overlay(x, y, ..., fun, filename="", datatype, format, overwrite, progress, recycle=TRUE)

## S4 method for signature 'Raster,missing'
overlay(x, y, ..., fun, filename="", datatype, format, overwrite, progress, unstack=TRUE)
```

Arguments

x	Raster* object
y	Raster* object, or missing (only useful if x has multiple layers)
...	Additional Raster objects
fun	Function to be applied. When using RasterLayer objects, the number of arguments of the function should match the number of Raster objects, or it should take any number of arguments. When using mult-layer objects the function should match the number of layers of the RasterStack/Brick object (unless unstack=FALSE)
filename	Character. Output filename (optional)
format	Character. Output file type. See writeFormats
datatype	Character. Output data type. See dataType
overwrite	Logical. If TRUE, "filename" will be overwritten if it exists
progress	Character. "text", "window", or "" (the default, no progress bar)
recycle	Logical. Should layers from Raster objects with fewer layers be recycled?
unstack	Logical. Should layers be ustacked before computation (i.e. does the fun refer to individual layers in a multilayer object)?

Details

Instead of the overlay function you can also use aritmethic functions such as `*, /, +, -` with Raster objects (see examples). In that case you cannot specifiy an output filename. Moreover, the overlay function should be more efficient when using large data files that cannot be loaded into memory, as the use of the complex arithmetic functions might lead to the creation of many temporary files.

While you can supply functions such as `sum` or `mean`, it would be more direct to use the Raster* objects are arguments to those functions (e.g. `sum(r1,r2,r3)`)

See `rasterize` and `extract` for "overlays" involving Raster* objects and polygons, lines, or points.

Value

Raster* object

Author(s)

Robert J. Hijmans

See Also

`calc`, `Arith-methods`

Examples

```
r <- raster(ncol=10, nrow=10)
r1 <- init(r, fun=runif)
r2 <- init(r, fun=runif)
r3 <- overlay(r1, r2, fun=function(x,y){return(x+y)})

# long version for multiplication
r4 <- overlay(r1, r2, fun=function(x,y){(x*y)} )

#use the individual layers of a RasterStack to get a RasterLayer
s <- stack(r1, r2)
r5 <- overlay(s, fun=function(x,y) x*y )
# equivalent to
r5c <- calc(s, fun=function(x) x[1]*x[2] )


#Combine RasterStack and RasterLayer objects (s2 has four layers. r1 (one layer) and s (two layers) are recycled)
s2 <- stack(r1, r2, r3, r4)
b <- overlay(r1, s, s2, fun=function(x,y,z){return(x*y*z)} )

# use a single RasterLayer (same as calc function)
r6 <- overlay(r1, fun=sqrt)

# multiplication with more than two layers (make sure the number of RasterLayers matches the arguments of 'fun'
r7 <- overlay(r1, r2, r3, r4, fun=function(a,b,c,d){return(a*b+c*d)} )
# equivalent function, efficient if values can be loaded in memory
r8 <- r1 * r2 + r3 * r4
```

```
# Also works with multi-layer objects.
s1 <- stack(r1, r2, r3)
x <- overlay(s1, s1, fun=function(x,y)x+y+5)

# in this case the first layer of the shorter object is recycled.
# i.e., s2 is treated as stack(r1, r3, r1)
s2 <- stack(r1, r3)
y <- overlay(s1, s2, fun=sum)
```

`pairs` *Pairs plot (matrix of scatterplots)*

Description

Pair plots of layers in a RasterStack or RasterBrick. This is a wrapper around graphics function `pairs`.

Usage

```
## S4 method for signature 'RasterStackBrick'
pairs(x, hist=TRUE, cor=TRUE, use="pairwise.complete.obs", maxpixels=100000, ...)
```

Arguments

`x`	RasterBrick or RasterStack
`hist`	Logical. If TRUE a histogram of the values is shown on the diagonal
`cor`	Logical. If TRUE the correlation coefficient is shown in the upper panels
`use`	Argument passed to the `cor` function
`maxpixels`	Integer. Number of pixels to sample from each layer of large Raster objects
`...`	Additional arguments (only `cex` and `main`)

Author(s)

Robert J. Hijmans

See Also

`boxplot`, `hist`, `density`

Examples

```
r <- raster(system.file("external/test.grd", package="raster") )
s <- stack(r, 1/r, sqrt(r))
pairs(s)

## Not run:
# to make indvidual histograms:
```

```
hist(r)
# or scatter plots:
plot(r, 1/r)

## End(Not run)
```

---

persp *Perspective plot*

Description

Perspective plot of a RasterLayer. This is an implementation of a generic function in the graphics package.

Usage

```
persp(x, ...)
```

Arguments

x	A Raster* object
...	Any argument that can be passed to persp (graphics package)

Methods

```
persp(x, y=1, z=NULL, ...)
```

x	a Raster* object
y	a index of x = RasterStack
z	values of z are ignored
...	

Author(s)

Robert J. Hijmans

See Also

persp, contour, plot

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
persp(r)
```

plot *Plot a Raster* object*

Description

Plot (that is, make a map of) the values of a Raster* object, or make a scatterplot of their values.

Points, lines, and polygons can be drawn on top of a map using `plot(..., add=TRUE)`, or functions like `points, lines, polygons`

`text` plots a textual (rather than color) representation of values of the cells on top of the cells.

Details

Most of the code for this function was taken from image.plot (fields package). Raster objects with a color-table (e.g. a graphics file) will be plotted according to the color table.

Methods

x=RasterLayer

```
plot(x, col=rev(terrain.colors(25)), maxpixels=100000, axes = TRUE, xlab="",
ylab="", alpha=1, useRaster=TRUE, ...)
```

`x` - RasterLayer object

`col` - A color palette, i.e. a vector of n contiguous colors such as rainbow, heat.colors, and topo.colors, or one or your own making, perhaps using `colorRampPalette`

`maxpixels` - Maximum number of pixels used for the map

`alpha` - Number between 0 and 1 to set transparancy. 0 is entirely transparant, 1 is not transparant

`useRaster` - If `TRUE`, the rasterImage function is used for plotting. Otherwise the image function is used. This can be useful if rasterImage does not work well on your system

`...` - Any argument that can be passed to `image.plot` and to `plot`

If `x` has a 'Color table', the default all to `plot` is:

```
plot(x, maxpixels=500000, ext=NULL, interpolate=FALSE, axes=FALSE, xlab="",
ylab="", ...)
```

`ext`	An extent object to zoom in a region
`interpolate`	Logical. Should the image be interpolated?
`axes`	Logical. Should axes be drawn?
`addfun`	Function. functions adding additional items such as points or polygons to the plot (map). Typically contain
`...`	Graphical parameters

x=RasterStack or RasterBrick, y=numeric or missing

```
plot(x, col=rev(terrain.colors(25)), subsample=TRUE, maxpixels=100000, alpha=1,
axes=TRUE, xlab="", ylab="", ...)
```

`x`	a RasterStack or RasterBrick object

y	The index of a RasterLayer in a RasterStack to be plotted. Either a single number between 1 and nl
other arguments	See above

x=Raster, y=Raster Produces a scatter plot of the values of x against those of y. `plot(x, y, maxpixels=100000, cex=0.1, ...)`

x	a RasterLayer object
y	a RasterLayer object
maxpixels	Maximum number of pixels in the map display
cex	point size
...	Any argument that can be passed to graphics::plot

x=Extent, y="ANY" Draws the (rectangular) bounding box of a raster object. `plot(ext)`

ext	a Extent object

Author(s)

Robert J. Hijmans

See Also

spplot, plotRGB, persp, contour, pairs, hist

The rasterVis package has more advanced plotting methods for Raster* objects.

Examples

```
# RasterLayer
r <- raster(nrows=10, ncols=10)
r <- setValues(r, 1:ncell(r))
plot(r)
text(r)

e <- extent(r)
plot(e, add=TRUE, col='red', lwd=4)
e <- e / 2
plot(e, add=TRUE, col='red')


# Scatterplot of 2 RasterLayers
r2 <- sqrt(r)
plot(r, r2)

# Multi-layer object (RasterStack / Brick)
s <- stack(r, r2, r/r)
plot(s, 2)
```

```
plot(s)

# two objects, different range, one scale:
r[] <- runif(ncell(r))
r2 <- r/2
brks <- seq(0, 1, by=0.1)
nb <- length(brks)-1
par(mfrow=c(1,2))
plot(r, breaks=brks, col=rev(terrain.colors(nb)), lab.breaks=brks, zlim=c(0,1))
plot(r2, breaks=brks, col=rev(terrain.colors(nb)), lab.breaks=brks, zlim=c(0,1))


# breaks and labels
x <- raster(nc=10, nr=10)
x[] <- runif(ncell(x))
brk <- c(0, 0.25, 0.75, 1)
arg <- list(at=c(0.12,0.5,0.87), labels=c("Low","Med.","High"))
plot(x, col=terrain.colors(3), breaks=brk)
plot(x, col=terrain.colors(3), breaks=brk, axis.args=arg)
par(mfrow=c(1,1))

# color ramp
plot(x, col=colorRampPalette(c("red", "white", "blue"))(255))

# adding random points to the map
xy <- cbind(-180 + runif(10) * 360, -90 + runif(10) * 180)
points(xy, pch=3, cex=5)

# for SpatialPolygons do
# plot(pols, add=TRUE)

# adding random points to each map of each layer of a RasterStack
fun <- function() {
points(xy, cex=2)
points(xy, pch=3, col='red')
}
plot(s, addfun=fun)
```

---

plotRGB *Red-Green-Blue plot of a multi-layered Raster object*

Description

Make a Red-Green-Blue plot based on three layers (in a RasterBrick or RasterStack). Three layers (bands) are combined such that one is the red channel, one is the green channel, and one is the blue channel. This function could be used to make 'True (or false) color images' from Landsat and other multi-band satellite images.

Methods

```
plotRGB(x, r=1, g=2, b=3, scale, maxpixels=500000, stretch=NULL, ext=NULL,
...)
```

x	a RasterBrick or RasterStack object
r	Integer. Index of the Red channel, between 1 and nlayers(x)
g	Integer. Index of the Green channel, between 1 and nlayers(x)
b	Integer. Index of the Blue channel, between 1 and nlayers(x)
scale	Integer. Maximum (possible) value in the three channels. Defaults to 255 or to the maximum value of x if tha
maxpixels	Maximum number of pixels to use
stretch	Option to stretch the values to increase the contrast of the image: "lin" or "hist"
ext	An extent object to zoom in a region
...	graphical parameters as in rasterImage

Author(s)

Robert J. Hijmans; stretch option based on functions by Josh Gray

See Also

plot

Examples

```
b <- brick(system.file("external/rlogo.grd", package="raster"))
plotRGB(b)
plotRGB(b, 3, 2, 1)
plotRGB(b, 3, 2, 1, stretch='hist')
```

---

pointDistance *Distance between points*

Description

Calculate the geographic distance between two (sets of) points on a sphere (longlat=TRUE) or on a plane (longlat=FALSE).

Usage

```
pointDistance(p1, p2, longlat, ...)
```

Arguments

p1	x and y coordinate of first (set of) point(s), either as c(x, y), matrix(ncol=2), or SpatialPoints*.
p2	x and y coordinate of second (set of) second point(s) (like for p1). If this argument is missing, a distance matrix is computed for p1
longlat	Logical. If TRUE, coordinates should be in degrees; else they should represent planar ('Euclidean') space (e.g. units of meters)
...	Additional arguments. Can be used to set the radius, r, of the world (modeled as a sphere), when longlat=TRUE Default is r=6378137

Value

A single value, or a vector, or matrix of values giving the distance in meters (longlat=TRUE) or map-units (for instance, meters in the case of UTM) If p2 is missing, a distance matrix is returned

Author(s)

Robert J. Hijmans and Jacob van Etten

See Also

distanceFromPoints, distance, gridDistance, spDistsN1

Examples

```
a <- cbind(c(1,5,55,31),c(3,7,20,22))
b <- cbind(c(4,2,8,65),c(50,-90,20,32))

pointDistance(c(0, 0), c(1, 1), longlat=FALSE)
pointDistance(c(0, 0), c(1, 1), longlat=TRUE)
pointDistance(c(0, 0), a, longlat=TRUE)
pointDistance(a, b, longlat=TRUE)

#Make a distance matrix
dst <- pointDistance(a, longlat=TRUE)
# coerce to dist object
dst <- as.dist(dst)
```

---

predict *Spatial model predictions*

Description

Make a Raster* object with predictions based on a a fitted model object. The first argument is a Raster* object with the independent variables. The layerNames in the RasterStack should exactely match those expected by the model. This will be the case if the same Raster* object was used (via extract) to obtain the values to fit the model (see the example). Any type of model (e.g. glm. gam, randomForest) for which a predict method has been implemented (or can be implemented) can be used.

Usage

```
## S4 method for signature 'Raster'
predict(object, model, filename="", fun=predict, ext=NULL, const=NULL, index=1, na.rm=TRUE, format, da
```

Arguments

object	Raster* object. Typcially a multi-layer type (RasterStack or RasterBrick)
model	A fitted model of any class that has a 'predict' method (or for which you can supply a similar method as fun argument. E.g. glm, gam, or randomForest
filename	Optional output filename
fun	Function. Default value is 'predict', but can be replaced with e.g. predict.se (depending on the type of model), or your own custom function.
ext	An Extent object to limit the prediction to a sub-region of x
const	data.frame. Can be used to add a constant for which there is no Raster object for model predictions. Particulalry useful if the constant is a character-like factor value for which it is currently not possible to make a RasterLayer
index	Integer. To select the column if predict.'model' returns a matrix with multiple columns
na.rm	Logical. Remove cells with NA values in the predictors before solving the model (and return a NA value for those cells). This option prevents errors with models that cannot handle NA values. In most other cases this will not affect the output. An exception is when predicting with a boosted regression trees model because these return predicted value even if some (or all!) variables are NA
format	Character. Output file type. See writeRaster (optional)
datatype	Character. Output data type. See dataType (optional)
overwrite	Logical. If TRUE, "filename" will be overwritten if it exists
progress	Character. "text", "window", or "" (the default, no progress bar)
...	Additional arguments to pass to the predict.'model' function

Value

RasterLayer or RasterBrick

Author(s)

Robert J. Hijmans

See Also

Use interpolate if your model has 'x' and 'y' as implicit independent variables (e.g., in kriging).

Examples

```
# A simple model to predict the location of the R in the R-logo using 20 presence points
# and 50 (random) pseudo-absence points. This type of model is often used to predict species distributions
# see the dismo package for more of that.

# create a RasterStack or RasterBrick with with a set of predictor layers
logo <- brick(system.file("external/rlogo.grd", package="raster"))
layerNames(logo)

## Not run:
# the predictor variables
par(mfrow=c(2,2))
plotRGB(logo, main='logo')
plot(logo, 1, col=rgb(cbind(0:255,0,0), maxColorValue=255))
plot(logo, 2, col=rgb(cbind(0,0:255,0), maxColorValue=255))
plot(logo, 3, col=rgb(cbind(0,0,0:255), maxColorValue=255))
par(mfrow=c(1,1))

## End(Not run)

#get presence and pseudo-absence points
p <- matrix(c(48, 48, 48, 53, 50, 46, 54, 70, 84, 85, 74, 84, 95, 85, 66, 42, 26, 4, 19, 17, 7, 14, 26, 29, 39, 45, 51,
#
a <- cbind(runif(250)*(xmax(logo)-xmin(logo))+xmin(logo), runif(250)*(ymax(logo)-ymin(logo))+ymin(logo))

#extract values for points from stack
xy <- rbind(cbind(1, p), cbind(0, a))
v <- data.frame(cbind(xy[,1], extract(logo, xy[,2:3])))
colnames(v)[1] <- 'pa'

#build a model, here an example with glm
model <- glm(formula=pa~., data=v)

#predict to a raster
r1 <- predict(logo, model, progress='window')

plot(r1)
points(p, bg='blue', pch=21)
points(a, bg='red', pch=21)

# use a modified function to get a RasterBrick with p and se
# from the glm model. The values returned by 'predict' are in a list,
# and this list needs to be transformed to a matrix

predfun <- function(model, data) {
  v <- predict(model, data, se.fit=TRUE)
  cbind(p=as.vector(v$fit), se=as.vector(v$se.fit))
}

# predfun returns two variables, so use index=1:2
r2 <- predict(logo, model, fun=predfun, index=1:2)
```

```
# principal components of a RasterBrick
# here using sampling to simulate an object too large
# too feed all its values to prcomp
sr <- sampleRandom(logo, 100)
pca <- prcomp(sr)

# note the use of the 'index' argument
x <- predict(logo, pca, index=1:3)
plot(x)

## Not run:
require(randomForest)
rfmod <- randomForest(pa ~., data=v)

## note the additional argument "type='response'" that is passed to predict.randomForest
r3 <- predict(logo, rfmod, type='response', progress='window')

## get a RasterBrick with class membership probabilities
v$pa <- as.factor(v$pa)
rfmod2 <- randomForest(pa ~., data=v)
r4 <- predict(logo, rfmod2, type='prob', index=1:2)
spplot(r4)


## End(Not run)
```

`Programming` *Helper functions for programming*

Description

These are low level functions that can be used by programmers to develop new functions. If in doubt, it is almost certain that you do not need these as these functions are already embedded in all other functions in the raster package.

`canProcessInMemory` is typically used within functions. In the raster package this function is used to determine if the amount of memory needed for the function is available. If there is not enough memory available, the function returns `FALSE`, and the function that called it will write the results to a temporary file.

openConnection opens a file connection for reading, closeConnection removes it.

pbCreate creates a progress bar, pbStep sets the progress, and pbClose closes it.

Usage

```
canProcessInMemory(x, n=4)
closeConnection(x)
openConnection(x, silent=FALSE)
pbCreate(nsteps, progress, style=3, ...)
```

```
pbStep(pb, step=NULL, label='')
pbClose(pb, timer)
getCluster()
returnCluster()
```

Arguments

x	RasterLayer or RasterBrick object
n	Integer. The number of copies of a RasterLayer object with values that a function needs to store in memory
silent	Logical. passed on to GDAL.open
nsteps	Integer. Number of steps the progress bar will make from start to end (e.g. nrow(raster))
progress	Character. 'text', 'window', or ''
style	style for text progress bar. See txtProgressBar
...	additinal arguments
pb	a progress bar object created with pbCreate
step	which step is this (1 <= step <= nsteps). If step is NULL, a single step is taken
label	label for the non-text progress bars
timer	Logical. If TRUE, time to completion will be printed. If missing, the value will be taken from the rasterOptions

Value

canProcessInMemory: logical

closeHandle: RasterLayer object

getCluster: snow cluster object

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(nrow=100, ncol=100)
canProcessInMemory(r, 4)
r <- raster(nrow=100000, ncol=100000)
canProcessInMemory(r, 2)
```

projection *Get or set a projection*

Description

Get or set the projection of a Raster* object

Usage

```
projection(x, asText = TRUE)
projection(x) <- value
```

Arguments

x	A Raster* object
asText	logical. If TRUE, the projection is returned as text. Otherwise a CRS object is returned
value	a CRS object or a character string describing a projection and datum in PROJ4 format

Details

projections are done by with the PROJ.4 library exposed by rgdal

Value

a Raster* object

Author(s)

Robert J. Hijmans

See Also

projectRaster, CRS-class, spTransform-methods, projInfo

Examples

```
r <- raster()
projection(r)
projection(r) <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"
projection(r)
```

`projectRaster` *Project a Raster object*

Description

Project the values of a Raster* object to a new Raster* object with another projection (coordinate reference system, (CRS)). You can do this by providing the new projection as a single argument in which case the function sets the extent and resolution of the new object. To have more control over the transformation, and, for example, to assure that the new object lines up with other datasets, you can provide a Raster* object with the properties that the input data should be projected to.

`projecExtent` returns a RasterLayer with a projected extent, but without any values. This RasterLayer can then be adjusted (e.g. by setting its resolution) and used as a template `'to'` in `projectRaster`.

Usage

```
projectRaster(from, to, res, crs, method="bilinear", filename="", ...)
projectExtent(object, crs)
```

Arguments

`from`	Raster* object
`to`	Raster* object with the parameters to which 'from' should be projected
`res`	Single or (vector of) two numerics. To, optionally, set the output resolution if 'to' is missing
`crs`	Character or object of class 'CRS'. PROJ.4 description of the coordinate reference system. In projectRaster this is used to set the output CRS if 'to' is missing, or if 'to' has no valid CRS
`method`	Method used to compute values for the new RasterLayer. Either 'ngb' (nearest neighbor), which is useful for categorical variables, or 'bilinear' (bilinear interpolation; the default value), which is appropriate for continuous variables.
`filename`	Character. Output filename
`...`	Additional arguments as for `writeRaster`
`object`	Raster* object

Details

There are two approaches you can follow to project the values of a Raster object.

1) Provide a `crs` argument, and, optionally, a `res` argument, but do not provide a `to` argument.

2) Create a template Raster with the CRS you want to project to. You can use an existing object, or use `projectExtent` for this or an existing Raster* object. Also set the number of rows and columns (or the resolution), and perhaps adjust the extent. The resolution of the output raster should normally be similar to that of the input raster. Then use that object as `from` argument to project the input Raster to. This is the preferred method because you have most control. For example you can assure that the resulting Raster object lines up with other Raster objects.

Projection is performed using the PROJ.4 library accesed through the rgdal package.

One of the best places to find PROJ.4 coordinate reference system descriptions is http://www.spatialreference.org.

You can also consult this page: http://www.remotesensing.org/geotiff/proj_list/ to find the parameter options and names for projections.

Also see `projInfo('proj')`, `projInfo('ellps')`, and `projInfo('datum')` for valid PROJ.4 values.

Value

RasterLayer or RasterBrick object.

Note

Vector (points, lines, polygons) can be transformed with `spTransform`.

`projecExtent` does not work very well when transforming projected circumpolar data to (e.g.) longitude/latitude. Which such data you may need to adjust the returned object. E.g. do `ymax(object) <- 90`

Author(s)

Robert J. Hijmans

See Also

`resample` `CRS-class`, `projInfo`, `spTransform`

Examples

```
# create a new (not projected) RasterLayer with cellnumbers as values
r <- raster(xmn=-110, xmx=-90, ymn=40, ymx=60, ncols=40, nrows=40)
r <- setValues(r, 1:ncell(r))
projection(r)
# proj.4 projection description
newproj <- "+proj=lcc +lat_1=48 +lat_2=33 +lon_0=-100 +ellps=WGS84"

# we need the rgdal package for this
if (require(rgdal)) {

#simplest approach
pr1 <- projectRaster(r, crs=newproj)

# alternatively also set the resolution
pr2 <- projectRaster(r, crs=newproj, res=20000)

# inverse projection, back to the properties of 'r'
inv <- projectRaster(pr2, r)

# to have more control, provide an existing Raster object, here we create one
# using projectExtent (no values are transferred)
```

```
pr3 <- projectExtent(r, newproj)
# Adjust the cell size
res(pr3) <- 200000
# now project
pr3 <- projectRaster(r, pr3)

## Not run:
# using a higher resolution
res(pr1) <- 10000
pr <- projectRaster(r, pr1, method='bilinear')
inv <- projectRaster(pr, r, method='bilinear')
dif <- r - inv
# small difference
plot(dif)

## End(Not run)

}
```

---

properties *Raster file properties*

Description

Properties of the values of the file that a RasterLayer object points to

`dataSize` returns the number of bytes used for each value (pixel, grid cell) `dataSigned` is TRUE for data types that include negative numbers.

Usage

```
dataSize(object)
dataSigned(object)
```

Arguments

`object`	Raster* object

Value

varies

Author(s)

Robert J. Hijmans

See Also

`filename`

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
dataSize(r)
dataSigned(r)
dataType(r)
```

---

quantile *Raster quantiles*

Description

Compute quantiles for the cell values of a RasterLayer. If you want to compute quantiles for each cell across a number of layers, you can use `calc(x, fun=quantile)`.

Usage

```
quantile(x, ...)
```

Arguments

x	Raster object
...	Additional arguments: `na.rm=TRUE`, `ncells=NULL`, and additional arguments to the stats::quantile function, see `quantile` ncells can be used to set the number of cells to be sampled, for very large raster datasets.

Value

A vector of quantiles

Author(s)

Robert J. Hijmans

See Also

`density`, `cellStats`

Examples

```
r <- raster(ncol=100, nrow=100)
r[] <- rnorm(ncell(r), 0, 50)
quantile(r)
quantile(r, probs = c(0.25, 0.75), type=7,names = FALSE)
```

raster *Create a RasterLayer object*

Description

Methods to create a RasterLayer object. RasterLayer objects can be created from scratch, a filename, an Extent object, a matrix, an 'image' object, or from a Raster*, Spatial*, asc, kasc (adehabitat*), grf (geoR) or kde object.

In many cases, e.g. when a RasterLayer is created from a file, it does (initially) not contain any cell (pixel) values, it only has the parameters that describe the RasterLayer.

For an overview of the functions in the raster package have a look here: raster-package.

Usage

```
raster(x, ...)
```

Arguments

x	Filename (character), Extent, Raster*, SpatialPixels*, SpatialGrid* object, matrix, or missing
...	Additional arguments, see below

Details

A new RasterLayer object normally has no cell-values in memory. If it is created from a file on disk, you can access cell-values with getValues, extract and related functions. You can assign new values with setValues and with replacement.

Value

RasterLayer object

Methods

1) Create a RasterLayer object from a file

```
raster(x, band=1, ...)
```

x	character. Name of raster file
band	integer. Band number in case of a file of multiple bands, default = 1
...	additional arguments.

Additional arguments:

native	Logical. Default is FALSE except when package rgdal is missing. If TRUE, reading and writing of IDRISI, BIL,
offset	Integer. To indicated the number of header rows on non-standard ascii files (rarely useful; use with caution)

For netCDF files (CF convention):

varname	character. The variable name (e.g. 'altitude' or 'precipitation'. If not supplied and the file has multiple variables
band	integer > 0. The 'band' (layer) number of the file. E.g., the 'time' variable (if there are any) (default=NA)

To read netCDF files, the ncdf package needs to be available.

If x is a character value, it should be a filename of a file that the raster package can read. Supported file types are the 'native' raster package format and those that can be read via rgdal. See readGDAL help for supported file types.

2) Create a RasterLayer object from scratch

```
raster(nrows=180, ncols=360, xmn=-180, xmx=180, ymn=-90, ymx=90, crs, ext)
```

nrows	number of rows
ncols	number of columns
xmn	minimum x coordinate (left border)
xmx	maximum x coordinate (right border)
ymn	minimum y coordinate (bottom border)
ymx	maximum y coordinate (top border)
crs	Character or object of class CRS. PROJ4 type description of a Coordinate Reference System (map projection). If t
ext	object of class Extent. If present, the arguments xmn, xmx, ymn and ynx are ignored

(item x is 'missing' in this case)

3) Create a RasterLayer object from an Extent-class object

```
raster(x, nrows=10, ncols=10, crs=NA)
```

x	Extent object
nrows	number of rows
ncols	number of columns
crs	PROJ4 type description of a map projection

4) Create a RasterLayer object from a Raster* object.

This copies the parameters of a Raster* object to a new RasterLayer, but does not copy the filename nor the cell values associated with the original Raster* object.

```
raster(x)
```

x	a Raster* object

5) Create a RasterLayer object from a RasterStack or RasterBrick object.

```
raster(x, layer=0)
```

x	a RasterStack, SpatialPixels* or SpatialGrid* object
layer	Integer. The layer from which to copy values to the new RasterLayer, if layer>0.

6) Create a RasterLayer object from a RasterStack, RasterBrick, SpatialPixels* or SpatialGrid* object.

```
raster(x, layer=0)
```

x	RasterStack, RasterBrick, SpatialPixels* or SpatialGrid* object
layer	Integer. the layer from which to copy values to the new RasterLayer, if layer>0

7) Create a RasterLayer object from a matrix.

The default extent is set to be between 0 and 1 in the x and y direction but can be changed at creation of the RasterLayer object or later. You can also provide a projection.

```
function(x, xmn=0, xmx=1, ymn=0, ymx=1, crs=NA)
```

x	matrix
xmn	minimum x coordinate (left border)
xmx	maximum x coordinate (right border)
ymn	minimum y coordinate (bottom border)
ymx	maximum y coordinate (top border)
crs	PROJ4 type description of a map projection (optional)

8) Create a RasterLayer object from an 'image' object (a list with vectors x and y and matrix z), or from an asc, kasc, or kde object

```
raster(img, crs)
```

x	RasterStack, SpatialPixels* or SpatialGrid* object
crs	PROJ4 type description of a map projection (optional)

Author(s)

Robert J. Hijmans

See Also

stack, brick

Examples

```
# Create a RasterLayer object from a file
#   N.B.: For your own files, omit the 'system.file' and 'package="raster"' bits
#   these are just to get the path to files installed with the package

f <- system.file("external/test.grd", package="raster")
f
r <- raster(f)

logo <- raster(system.file("external/rlogo.grd", package="raster"))
```

```
#from scratch
r1 <- raster(nrows=108, ncols=21, xmn=0, xmx=10)

#from an Extent object
e <- extent(r)
r2 <- raster(e)

#from another Raster* object
r3 <- raster(r)
s <- stack(r, r, r)
r4 <- raster(s)
r5 <- raster(s, 3)
```

Raster-class *Raster* classes*

Description

A raster is a database organized as a rectangular grid that is sub-divided into rectangular cells of equal area (in terms of the units of the coordinate reference system). The 'raster' package defines a number of "S4 classes" to manipulate such data.

The main user level classes are `RasterLayer`, `RasterStack` and `RasterBrick`. They all inherit from `BasicRaster` and can contain values for the raster cells.

An object of the `RasterLayer` class refers to a single layer (variable) of raster data. The object can point to a file on disk that holds the values the raster cells, or hold these values in memory. Or it can not have any associated values at all.

A `RasterStack` represents a collection of `RasterLayer` objects with the same extent and resolution. Organizing `RasterLayer` objects in a `RasterStack` can be practical when dealing with multiple layers; for example to summarize their values (see `calc`) or in spatial modeling (see `predict`).

An object of class `RasterBrick` can also contain multiple layers of raster data, but they are more tightly related. An object of class `RasterBrick` can refer to only a single (multi-band) data file, whereas each layer in a `RasterStack` can refer to another file (or another band in a multi-band file). This has implications for processing speed and flexibility. A `RasterBrick` should process quicker than a `RasterStack` (irrespective if values are on disk or in memory). However, a `RasterStack` is more flexible as a single object can refer to layers that have values stored on disk as well as in memory. If a layer that does not refer to values on disk (they only exists in memory) is added to a `RasterBrick`, it needs to load all its values into memory (and this may not be possible because of memory size limitations).

Objects can be created from file or from each other with the following functions: `raster`, `brick` and stack.

`Raster*` objects can also be created from SpatialPixels* and SpatialGrid* objects from the sp package using `as`, or the simply with the function `raster`, `brick`, or `stack`. Vice versa, `Raster*` objects can be coerced into a sp type object with `as( , )`, e.g. `as(x, 'SpatialGridDataFrame')` .

Common generic methods implemented for these classes include:

summary, show, dim, and plot, ...

[is implemented for RasterLayer.

The classes described above inherit from the BasicRaster class which inherits from BasicRaster. The BasicRaster class describes the main properties of a raster such as the number of columns and rows, and it contains an object of the link[raster]{Extent-class} to describe its spatial extent (coordinates). It also holds the 'coordinate reference system' in a slot of class CRS-class defined in the sp package. A BasicRaster cannot contain any raster cell values is therefore seldomly used.

The Raster* class inherits from BasicRaster. It is is a virtual class; which means that you cannot create an object of this class. It is used only to define methods for all the classes that inherit from it (RasterLayer, RasterStack and RasterBrick). Another virtual class is the RasterStackBrick class. It is formed by a class union of RasterStack and RasterBrick. You cannot make objects of it, but methods defined for objects of this class as arguments will accept objects of the RasterLayer and RasterStack as that argument.

Classes RasterLayer and RasterBrick have a slot with an object of class RasterFile that describes the properties of the file they point to (if they do). RasterLayer has a slot with an object of class SingleLayerData, and the RasterBrick class has a slot with an object of class MultipleLayerData. These 'datalayer' classes can contain (some of) the values of the raster cells

These classes are not further described here because users should not need to directly access these slots. The 'setter' functions such as setValues should be used instead. Using such 'setter' functions is much safer because a change in one slot should often affect the values in other slots.

Objects from the Class

Objects can be created by calls of the form new("RasterLayer", ...), or with the helper functions such as raster.

Slots

Slots for Raster* objects

title: Character

file: Object of class ".RasterFile"

data: Object of class ".SingleLayerData" or ".MulitpleLayerData". Not in RasterStack objects

history: To record processing history, not yet in use

legend: Object of class .RasterLegend, Default legend. Should store preferences for plotting. Not yet implemented except that it stores the color table of images, if available

extent: Object of Extent-class

ncols: Integer

nrows: Integer

crs: Object of class "CRS", i.e. the coordinate reference system. In Spatial* objects this slot is called 'proj4string'

Author(s)

Robert J. Hijmans

Examples

```
showClass("RasterLayer")
```

---

rasterFromCells *Subset a raster by cell numbers*

Description

This function returns a new raster based on an existing raster and cell numbers for that raster. The new raster is cropped to the cell numbers provided, and, if `values=TRUE` has values that are the cell numbers of the original raster.

Usage

```
rasterFromCells(x, cells, values=TRUE)
```

Arguments

`x`	Raster* object (or a SpatialPixels* or SpatialGrid* object)
`cells`	vector of cell numbers
`values`	Logical. If `TRUE`, the new RasterLayer has cell values that correspond to the cell numbers of `x`

Details

Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom The last cell number equals the number of cells of the Raster* object.

Value

RasterLayer

Author(s)

Robert J. Hijmans

See Also

`rowFromCell`

Examples

```
r <- raster(ncols=100, nrows=100)
cells <- c(3:5, 210)
r <- rasterFromCells(r, cells)
cbind(1:ncell(r), getValues(r))
```

rasterFromXYZ *Create a RasterLayer from x, y, z values*

Description

Create a RasterLayer from x, y, and z values. x and y must be on a regular grid. If the resolution is not supplied, it is assumed to be the minimum distance between x and y coordinates, but a resolution of up to 10 times smaller is evaluated if a regular grid can otherwise not be created. If the exact properties of the RasterLayer are known beforehand, it may be preferable to simply create a new RasterLayer with the raster function instead, compute cell numbers and assign the values with these (see example below).

Usage

```
rasterFromXYZ(xyz, res=c(NA,NA), crs=NA, digits=5)
```

Arguments

xyz	Matrix or data.frame with three colomns: x and y coordinates, and value z
res	The x and y cell resolutin (optional)
crs	A CRS object or a character string describing a projection and datum in PROJ.4 format
digits	Precision for detecting wether points are on a regular grid (i.e., a low number of digits is a low precision)

Value

RasterLayer

Author(s)

Robert J. Hijmans

See Also

For random-like point distributions, see rasterize

Examples

```
r <- raster(nrow=10, ncol=10, xmn=0, xmx=10, ymn=0, ymx=10, crs=NA)
r[] <- runif(ncell(r))
r[r<0.5] <- NA
xyz <- rasterToPoints(r)

r2 <- rasterFromXYZ(xyz)

# equivalent to:
r3 <- raster(nrow=10, ncol=10, xmn=0, xmx=10, ymn=0, ymx=10)
```

```
cells <- cellFromXY(r3, xyz[,1:2])
r3[cells] <- xyz[,3]
```

`rasterize` *Rasterize points, lines, or polygons*

Description

Transfer values associated with 'vector' type spatial data (points, lines, polygons) to the spatially overlapping raster cells.

For polygons, 'spatially overlapping' means that the polygon must cover the center of a raster cell. All cells that are touched by a line are considered to be overlapping. Points that fall on a border between cells are placed in the cell to the right and/or in the cell below.

Usage

```
rasterize(x, y, ...)
```

Arguments

`x`	points (a SpatialPoints* object, or a two-column matrix), SpatialLines*, SpatialPolygons*, or an Extent object
`y`	Raster* object
`...`	Additional arguments, see under Details

Value

RasterLayer or RasterBrick

Methods

= all cases =

If `x` is a Spatial*DataFrame, the index (integer), or column name (character) of the variable to be transfered. If `field < 0`, all features get the attribute value 1, and if `field == 0`, the attribute index is used (i.e. numbers from 1 to the number of features). If the Spatial object has a data.frame, all values >= 1 will use the attribute index. In all cases you can also provide a vector with the same length as the number of spatial features, or a matrix where the number of rows matches the number of spatial features

`field` Determine what values to assign to cells that are covered by multiple spatial features. You can use functions such as `min`, `max`, or `mean`, or one of the following character values: `'first'`, `'last'`, `'sum'`, `'min'`, or `'max'`. The default value is `'last'`

`background` Value to put in the cells that are not covered by any of the features of `x`. Default is `NA`

`mask` Logical. If `TRUE` the values of the input Raster object are 'masked' by the spatial features of `x`. That is, cells that spatially overlap with the spatial features retain their values, the other cells become `NA`. Default is `FALSE`. This option cannot be used when `update=TRUE`

`update` Logical. If `TRUE`, the values of the Raster* object are updated for the cells that overlap the spatial features of `x`. Default is `FALSE`. Cannot be used when `mask=TRUE`

`updateValue` Numeric (normally an integer), or character. Only relevant when `update=TRUE`. Select, by their values, the cells to be updated with the values of the spatial features. Valid character values are `'all'`, `'NA'`, and `'!NA'`. Default is `'all'`

`filename` Character. Output filename (optional)

`overwrite` Logical. If TRUE, "filename" will be overwritten if it exists

`format` Character. Output file type. See writeRaster

`datatype` Character. Output data type. See dataType

`progress` Character. "text", "window", or "" (the default, no progress bar)

= points =

The value of a grid cell is determined by the values associated with the points and function `fun`.

If `x` represents points, each point is assinged to a grid cell. The default result is that cells with one or more points get value 1 and the other cells get the background value (default is `NA`).

If you want to know the number of points in each grid cell, use `fun=function(x,...){length(x)}`. I.e., for each cell it computes the length of the vector of points. You can also use `fun=sum`, because the default 'field' value of a point is 1. For the number of unique values use `fun=function(x, ...){ length(unique(na.rm(x)))}`

There is one additional argument:

`na.rm` If `TRUE`, `NA` values are removed if `fun` honors the `na.rm` argument

Because of this argument, all functions `fun` must accept an `na.rm` argument, either explicitly or through 'dots'. This means that `fun=length` fails, but `fun=function(x,...)length(x)` works, although it ignores the `na.rm` argument. To use the `na.rm` argument you can use a function like this `fun=function(x,na.rm){if (na.rm) length(na.omit(x)) else (length(x)}`, or use a function that removes NA values in all cases, like this function to compute the number of unique values "richness": `fun=function(x, ...){length(unique(na.omit(x)))}`.

You can also pass multiple functions using a statement like `fun=function(x)c(lenght(x),mean(x))`, in which case the returned object is a RasterBrick (multiple layers).

= polygons and Extent =

A polygon value is transferred to a raster-cell if it covers the center of the cell. Either values associated with each polygon, or a polygon ID is transferred. Holes in polygons are recognized if they are correctly specified. The following additional arguments are available:

`getCover` Logical. If `TRUE`, the fraction of each grid cell that is covered by the polygons is returned (and the values of `field`, `fun`, `mask`, and `update` are ignored. The fraction covered is estimated by dividing each cell into 100 subcells and determining presence/absence of the polygon in the center of each subcell

`silent` Logical. If `TRUE`, feedback on the polygon count is suppressed. Default is `FALSE`

Author(s)

Robert J. Hijmans

See Also

`extract`

Examples

```
################################
# rasterize points
################################
r <- raster(ncols=36, nrows=18)
n <- 1000
x <- runif(n) * 360 - 180
y <- runif(n) * 180 - 90
xy <- cbind(x, y)
# point or not?
r1 <- rasterize(xy, r)
# how many points?
r2 <- rasterize(xy, r, fun=sum)
vals <- runif(n)
# sum of the values associated with the points
r3 <- rasterize(xy, r, vals, fun=sum)

# with a SpatialPointsDataFrame
vals <- 1:n
p <- as.data.frame(cbind(xy, name=vals))
coordinates(p) <- ~x+y
r <- rasterize(p, r, 'name', fun=min)
#r2 <- rasterize(p, r, 'name', fun=max)
#plot(r, r2, cex=0.5)

################################
# rasterize lines
################################
cds1 <- rbind(c(-180,-20), c(-140,55), c(10, 0), c(-140,-60))
cds2 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55))
cds3 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45))

lines <- SpatialLines(list(Lines(list(Line(cds1)), "1"), Lines(list(Line(cds2)), "2"), Lines(list(Line(cds3)), "3

r <- raster(ncols=90, nrows=45)
r <- rasterize(lines, r)

## Not run:
plot(r)
plot(lines, add=TRUE)

r <- rasterize(lines, r, fun='count')
plot(r)

r[] <- 1:ncell(r)
r <- rasterize(lines, r, mask=TRUE)
plot(r)

r[] <- 1
r[lines] <- 10
plot(r)
```

```
## End(Not run)

##############################
# rasterize polygons
##############################

p1 <- rbind(c(-180,-20), c(-140,55), c(10, 0), c(-140,-60), c(-180,-20))
hole <- rbind(c(-150,-20), c(-100,-10), c(-110,20), c(-150,-20))
p2 <- rbind(c(-10,0), c(140,60), c(160,0), c(140,-55), c(-10,0))
p3 <- rbind(c(-125,0), c(0,60), c(40,5), c(15,-45), c(-125,0))
pols <- SpatialPolygons( list( Polygons(list(Polygon(p1), Polygon(hole)), 1), Polygons(list(Polygon(p2)), 2), Pol
pols@polygons[[1]]@Polygons[[2]]@hole <- TRUE

r <- raster(ncol=180, nrow=90)
r <- rasterize(pols, r, fun='sum')

## Not run:
plot(r)
plot(pols, add=T)

# add a polygon
p5 <- rbind(c(-180,10), c(0,90), c(40,90), c(145,-10),  c(-25, -15), c(-180,0), c(-180,10))
addpoly <- SpatialPolygons(list(Polygons(list(Polygon(p5)), 1)))
addpoly <- as(addpoly, "SpatialPolygonsDataFrame")
addpoly@data[1,1] <- 10
r2 <- rasterize(addpoly, r, field=1, update=TRUE, updateValue="NA")
plot(r2)
plot(pols, border="blue", lwd=2, add=TRUE)
plot(addpoly, add=TRUE, border="red", lwd=2)

# get the percentage cover of polygons in a cell
r3 <- raster(ncol=36, nrow=18)
r3 <- rasterize(pols, r3, getCover=TRUE)

## End(Not run)
```

`rasterTmpFile` *Temporary files*

Description

Functions in the raster package create temporary files if the values of an output RasterLayer cannot be stored in memory (ram). This can happen when no filename is provided to a function and in functions where you cannot provide a filename (e.g. when using 'raster algebra').

Temporary files are automatically removed at the start of each session. During a session you can use `showTmpFiles` to see what is there and `removeTmpFiles` to delete all the temporary files. `rasterTmpFile` returns a temporary filename. These can be useful when developing your own functions.

Usage

```
rasterTmpFile(prefix='raster_tmp_')
showTmpFiles()
removeTmpFiles(h=24)
```

Arguments

prefix	Character. Prefix to the filename (which will be followed by 10 random numbers)
h	Numeric. The minimum age of the files in number of hours (younger files are not deleted)

Details

The default path where the temporary files are stored is returned by `showOptions()`; you can change it (for the current session) with setOptions.

Value

`rasterTmpFile` returns a valid file name

`showTmpFiles` returns the names (.grd only) of the files in the temp directory

`removeTmpFiles` returns nothing

Author(s)

Robert J. Hijmans

See Also

`setOptions`, `showOptions`, `tempfile`

Examples

```
#rasterTmpFile('mytemp_')
showTmpFiles()
removeTmpFiles(h=24)
```

---

rasterToContour *Raster to contour lines conversion*

Description

RasterLayer to contour lines. This is a wrapper around `contourLines`

Usage

```
rasterToContour(x, maxpixels=100000, ...)
```

Arguments

x	a RasterLayer object
maxpixels	Maximum number of raster cells to use; this function fails when too many cells are used
...	Any argument that can be passed to contourLines

Details

Most of the code was taken from maptools::ContourLines2SLDF, by Roger Bivand & Edzer Pebesma

Value

SpatialLinesDataFrame

Author(s)

Robert J. Hijmans

Examples

```
f <- system.file("external/test.grd", package="raster")
r <- raster(f)
x <- rasterToContour(r)
class(x)
plot(r)
plot(x, add=TRUE)
```

---

rasterToPoints *Raster to points conversion*

Description

Raster to point conversion. Cells with NA are not converted. A function can be used to select a subset of the raster cells (by their values).

Usage

```
rasterToPoints(x, fun=NULL, spatial=FALSE, ...)
```

Arguments

x	A Raster* object
fun	Function to select a subset of raster values
spatial	Logical. If TRUE, the function returns a SpatialPointsDataFrame object
...	Additional arguments. Currently only progress to specify a progress bar. "text", "window", or "" (the default, no progress bar)

Details

fun should be a simple function returning a logical value e.g.: `fun=function(x){x==1}` or `fun=function(x){x>3}`

Value

A matrix with three columns: x, y, and v (value), or a SpatialPointsDataFrame object

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(nrow=18, ncol=36)
r[] <- runif(ncell(r)) * 10
r[r>8] <- NA
p <- rasterToPoints(r)
p <- rasterToPoints(r, fun=function(x){x>6})
#plot(r)
#points(p)
```

---

rasterToPolygons *Raster to polygons conversion*

Description

Raster to polygons conversion. Cells with NA are not converted. A function can be used to select a subset of the raster cells (by their values).

Usage

```
rasterToPolygons(x, fun=NULL, n=4, na.rm=TRUE, digits=12, dissolve=FALSE)
```

Arguments

x	a Raster* object
fun	function to select a subset of raster values (only allowed if x has a single layer
n	The number of nodes for each polygon. Only 4, 8, and 16 are allowed
na.rm	If TRUE, cells with NA values in all layers are ignored
digits	number of digits to round the coordinates to
dissolve	logical. If TRUE, polygons with the same attribute value will be dissolved into multi-polygon regions. This option requires the rgeos package

Details

fun should be a simple function returning a logical value e.g.: `fun=function(x){x==1}` or `fun=function(x){x>3 & x<6}`

Value

SpatialPolygonsDataFrame

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(nrow=18, ncol=36)
r[] <- runif(ncell(r)) * 10
r[r>8] <- NA
pol <- rasterToPolygons(r, fun=function(x){x>6})
#plot(r)
#plot(pol, add=T, col='red')
```

---

readAll *Read values from disk*

Description

Read all values from a raster file associated with a Raster* object into memory. This function should normally not be used. In most cases `getValues` or `getValuesBlock` is more approprate as readAll will fail when there is no file associated with the RasterLayer (values may only exist on memory).

Usage

```
readAll(object)
```

Arguments

object	a Raster* object

Author(s)

Robert J. Hijmans

See Also

`getValues`, `getValuesBlock`, `extract`

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
r <- readAll(r)
```

`reclass` *Reclassify*

Description

Reclassify values of a Raster* object. The function (re)classifies groups of values to other values. E.g. All values between 1 and 10 become 1, and all values between 11 and 15 become 2.

Reclassification is done with matrix "rcl". For simple cases, the functions `subs` and `cut` may be more efficient. Reclassification is done in the order of the reclass table. Thus, if there are overlapping ranges, the last range applies.

Usage

```
## S4 method for signature 'Raster'
reclass(x, rcl, filename='', include.lowest=FALSE, right=TRUE, ...)
```

Arguments

`x`	Raster* object
`rcl`	matrix for reclassifcation. This matrix must have 3 columns. The first two columns are "from" "to" of the input values, and the third column "becomes" has the new value for that range. (You can also supply a vector that can be coerced into a n*3 matrix (with byrow=TRUE)). You can also provide a two column matrix ("is", "becomes") which can be useful for integer values. In that case, the `right` argument is automatically set to `NA`
`filename`	character. Output filename (optional)
`include.lowest`	logical, indicating if a value equal to the lowest value in rcl (or highest value in the second column, for right = FALSE) should be included. The default is `FALSE`
`right`	logical, indicating if the intervals should be closed on the right (and open on the left) or vice versa. The default is `TRUE`. A special case is to use right=NA. In this case both the left and right intervals are open
`...`	additional arguments as for `writeRaster`

Value

Raster* object

Author(s)

Robert J. Hijmans

See Also

subs, cut, calc

Examples

```
r <- raster(ncols=36, nrows=18)
r[] <- runif(ncell(r))
# reclassify the values into three groups
# all values >= 0 and <= 0.25 become 1, etc.
m <- c(0, 0.25, 1,  0.25, 0.5, 2,  0.5, 1, 3)
rclmat <- matrix(m, ncol=3, byrow=TRUE)
rc <- reclass(r, rclmat)

# equivalent to
rc <- reclass(r, c(-Inf,0.25,1, 0.25,0.5,2, 0.5,Inf,3))
```

---

rectify *rectify a Raster object*

Description

rectify changes a rotated Raster* object into a non-rotated (rectangular) object. This is wrapper function around resample.

Usage

```
rectify(x, ext, res, method='ngb', filename='', ...)
```

Arguments

x	Raster* object to be rectified
ext	Optional. Extent object or object from which an Extent object can be extracted
res	Optional. Single or two numbers to set the resolution
method	Method used to compute values for the new RasterLayer, should be "bilinear" for bilinear interpolation, or "ngb" for nearest neighbor
filename	Character. Output filename
...	Additional arguments as for writeRaster

Value

RasterLayer or RasterBrick object

Author(s)

Robert J. Hijmans

replacement *Replace cell values of a Raster* object*

Description

You can set values of a Raster* object, when `i` is a vector of cell numbers, a Raster*, Extent, or Spatial* object.

These are shorthand methods that only work for relatively small Raster* objects. In other cases you can use functions such as `calc` and `rasterize`.

Methods

`x[i] <- value`

`x[i,j] <- value`

Arguments

`x`	a Raster* object
`i`	cell number(s), row number(s), Extent, Spatial* object
`j`	columns number(s) (only available if i is (are) a row number(s))
`value`	new cell value(s)

See Also

calc, rasterize

Examples

```
r <- raster(ncol=10, nrow=5)
r[] <- 1:ncell(r) * 2
r[1,] <- 1
r[,1] <- 2
r[1,1] <- 3

s <- stack(r, sqrt(r))
s[s<5] <- NA
```

resample *Resample a Raster object*

Description

Resample transfers values between non matching Raster* objects (in terms of origin and resolution). Use `projectRaster` if the target has a different coordinate reference system (projection).

Before using resample, you may want to consider using these other functions instead: `aggregate`, `disaggregate`, `crop`, `expand`, `merge`.

Usage

```
## S4 method for signature 'Raster,Raster'
resample(x, y, method="bilinear", filename="", ...)
```

Arguments

x	Raster* object to be resampled
y	Raster* object with parameters that x should be resample to
method	method used to compute values for the new RasterLayer, should be "bilinear" for bilinear interpolation, or "ngb" for using the nearest neighbor
filename	character. Output filename (optional)
...	Additional arguments as for writeRaster

Value

RasterLayer or RasterBrick object

Author(s)

Robert J. Hijmans

See Also

aggregate, disaggregate, crop, expand, merge, projectRaster

Examples

```
r <- raster(nrow=3, ncol=3)
r[] <- 1:ncell(r)
s <- raster(nrow=10, ncol=10)
s <- resample(r, s, method='bilinear')
#par(mfrow=c(1,2))
#plot(r)
#plot(s)
```

---

resolution *Resolution*

Description

Get (or set) the x and/or y resolution of a Raster* object

Usage

```
xres(x)
yres(x)
res(x)
res(x) <- value
```

Arguments

x	Raster* object
value	Resolution (single number or vector of two numbers)

Value

A single numeric value or two numeric values.

Author(s)

Robert J. Hijmans

See Also

extent, ncell

Examples

```
r <- raster(ncol=18, nrow=18)
xres(r)
yres(r)
res(r)

res(r) <- 1/120
# set yres diffrently
res(r) <- c(1/120, 1/60)
```

---

rotate *Rotate*

Description

Rotate a Raster* object that has x coordinates (longitude) from 0 to 360, to standard coordinates between -180 and 180 degrees. Longitude between 0 and 360 is frequently used in data from global climate models.

Usage

```
rotate(x, ...)
```

Arguments

x	Raster* object
...	Additional arguments as for writeRaster

Value

RasterLayer or a RasterBrick object

Author(s)

Robert J. Hijmans

See Also

flip

Examples

```
r <- raster(nrow=18, ncol=36)
m <- matrix(1:ncell(r), nrow=18)
r[] <- as.vector(t(m))
extent(r) <- extent(0, 360, -90, 90)
rr <- rotate(r)
```

---

rotated *Do the raster cells have a rotation?*

Description

Do the raster cells have a rotation?

Usage

```
rotated(x)
```

Arguments

x	A Raster* object

Value

Logical value

Author(s)

Robert J. Hijmans

See Also

rectify

Examples

```
r <- raster()
rotated(r)
```

---

round *Integer values*

Description

These functions take a single RasterLayer argument x and change its values to integers.

ceiling returns a RasterLayer with the smallest integers not less than the corresponding values of x.

floor returns a RasterLayer with the largest integers not greater than the corresponding values of x.

trunc returns a RasterLayer with the integers formed by truncating the values in x toward 0.

round returns a RasterLayer with values rounded to the specified number of digits (decimal places; default 0).

Details

see ?base::round

Value

a RasterLayer object

Methods

ceiling(x) floor(x) trunc(x, ...) round(x, digits = 0)

a RasterLayer object

digits integer indicating the precision to be used

... additional arguments

Author(s)

Robert J.Hijmans

See Also

round

Examples

```
r <- raster(ncol=10, nrow=10)
r[] <- runif(ncell(r)) * 10
s <- round(r)
```

rowFromCell *Row or column number from a cell number*

Description

These functions get the row and/or column number from a cell number of a Raster* object)

Usage

```
colFromCell(object, cell)
rowFromCell(object, cell)
rowColFromCell(object, cell)
```

Arguments

object	Raster* object (or a SpatialPixels* or SpatialGrid* object)
cell	cell number(s)

Details

The colFromCell and similar functions accept a single value, or a vector or list of these values, Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom The last cell number equals the number of cells of the Raster* object.

Value

row of column number(s)

Author(s)

Robert J. Hijmans

See Also

cellFrom

Examples

```
r <- raster(ncols=10, nrows=10)
colFromCell(r, c(5,15))
rowFromCell(r, c(5,15))
rowColFromCell(r, c(5,15))
```

SampleInt *Sample integer values*

Description

Take a random sample from a range of integer values between 1 and n. Its purpose is similar to that of sample, but that function fails when n is very large.

Usage

```
sampleInt(n, size, replace=FALSE)
```

Arguments

n	Positive number (integer); the number of items to choose from
size	Non-negative integer; the number of items to choose
replace	Logical. Should sampling be with replacement?

Value

vector of integer numbers

Author(s)

Robert J. Hijmans

Examples

```
  sampleInt(1e+12, 10)

# this may fail:
#  sample.int(1e+12, 10)
#  sample.int(1e+9, 10)
```

sampleRandom *Random sample*

Description

Take a random sample from the cell values of a Raster* object (without replacement).

Usage

```
sampleRandom(x, size, ...)
```

Arguments

x	a Raster object
size	positive integer giving the number of items to choose.
...	Additional arguments: na.rm Logical. If TRUE (the default), NA values are removed from random sample ext Extent. To limit regular sampling to the area within the extent cells Logical. If TRUE, sampled cell numbers are also returned rowcol Logical. If TRUE, sampled row and column numbers are also returned sp Logical. If TRUE, a SpatialPointsDataFrame is returned asRaster Logical. If TRUE, a Raster* object is returned with random cells with values, all other cells with NA

Details

With na.rm=TRUE, the returned sample may be smaller than requested

Value

A vector, matrix (if cells=TRUE or x is a multi-layered object), or a SpatialPointsDataFrame (if sp=TRUE)

Author(s)

Robert J. Hijmans

See Also

sampleRegular

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
sampleRandom(r, size=10)
s <- stack(r, r)
sampleRandom(s, size=10, cells=TRUE, sp=TRUE)
```

---

sampleRegular *Regular sample*

Description

Take a systematic sample from a Raster* object.

Usage

```
sampleRegular(x, size, ext=NULL, cells=FALSE, asRaster=FALSE, useGDAL=FALSE)
```

Arguments

x	a Raster object
size	positive integer giving the number of items to choose.
ext	Extent. To limit regular sampling to the area within that box
cells	Logical. Also return sampled cell numbers (if asRaster=FALSE)
asRaster	Logical. If TRUE, a RasterLayer or RasterBrick is returned, rather then the sampled values
useGDAL	Logical. If TRUE, GDAL is used to sample in some cases. This is quicker, but can result in values for a different set of cells. Only for rasters that are accessed via rgdal, are not rotated, and when cells=FALSE.

Value

A vector, matrix (if cells=TRUE; or for a multi-layered object), or RasterLayer (if asRaster=TRUE)

Author(s)

Robert J. Hijmans

See Also

sampleRandom

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
v <- sampleRegular(r, size=100)
x <- sampleRegular(r, size=100, asRaster=TRUE)
```

---

saverasterstack *Save or open a RasterStack file*

Description

A RasterStack is a collection of RasterLayers with the same spatial extent and resolution. They can be crated from RasterLayer objects, or from file names. These two functions allow you to save the references to raster files and recreate a rasterStack object later. They only work if the RasterStack points to layers that have their values on disk. The values are not saved, only the references to the files.

Usage

```
stackOpen(stackfile)
stackSave(x, filename)
```

Arguments

stackfile	Filename for the RasterStack (to save it on disk)
x	RasterStack object
filename	File name

Details

When a RasterStack is saved to a file, only pointers (filenames) to raster datasets are saved, not the data. If the name or location of a raster file changes, the RasterStack becomes invalid.

Value

RasterStack object

Author(s)

Robert J. Hijmans

See Also

writeRaster, stack, addLayer

Examples

```
file <- system.file("external/test.grd", package="raster")
s <- stack(c(file, file))
s <- stackSave(s, "mystack")
# note that filename adds an extension .stk to a stackfile
## Not run:
s2 <- stackOpen("mystack.stk")
s2

## End(Not run)
```

---

scalebar *scalebar*

Description

Add a scalebar to a plot

Usage

```
scalebar(d, xy = NULL, type = "line", divs = 2, below = "", lonlat = NULL, label, adj=c(0.5, -0.5), lwd =
```

Arguments

d	distance covered by scalebar
xy	x and y coordinate to place the plot. Can be NULL. Use xy=click() to make this interactive
type	"line" or "bar"
divs	Number of divisions for a bar type. 2 or 4
below	Text to go below scalebar (e.g., "kilometers")
lonlat	Logical or NULL. If logical, TRUE indicates if the plot is using longitude/latitude coordinates. If NULL this is guessed from the plot's coordinates
adj	adjustment for text placement
label	Vector of three numbers to label the scale bar (beginning, midpoint, end)
lwd	line width for the "line" type scalebar
...	arguments to be passed to other methods

Value

None. Use for side effect of a scalebar added to a plot

Author(s)

Robert J. Hijmans; partly based on a function by Josh Gray

See Also

plot

Examples

```
r <- raster( system.file("external/test.grd", package="raster") )
plot(r)
scalebar(1000)
scalebar(1000, xy=c(178000, 333500), type='bar', divs=4)
```

---

setExtent *Set the extent of a RasteLayer*

Description

setExtent sets the extent of a Raster* object. Either by providing a new Extent object or by setting the extreme coordinates one by one.

Usage

```
setExtent(x, ext, keepres=FALSE, snap=FALSE)
extent(x) <- value
```

Arguments

x	A Raster* object
ext	An object of class Extent (which you can create with extent, or an object that has an extent (e.g. a Raster* or Spatial* object))
keepres	logical. If TRUE, the resolution of the cells will stay the same after adjusting the bounding box (by adjusting the number of rows and columns). If FALSE, the number of rows and columns will stay the same, and the resolution will be adjusted.
snap	logical. If TRUE, the extent is adjusted so that the cells of the input and output RasterLayer are aligned
value	An object of class Extent (which you can create with extent)

Value

a Raster* object

Author(s)

Robert J. Hijmans

See Also

extent, Extent-class

Examples

```
r <- raster()
bb <- extent(-10, 10, -20, 20)
extent(r) <- bb
r <- setExtent(r, bb, keepres=TRUE)
```

---

setMinMax *Compute min and max values*

Description

The minimum and maximum value of a RasterLayer are computed (from a file on disk if necessary) and stored in the returned Raster* object.

Usage

```
setMinMax(x)
```

Arguments

x	A Raster* object

Value

a Raster* object

Author(s)

Robert J. Hijmans

See Also

getValues

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
r
r <- setMinMax(r)
r
```

---

setValues *Set values of a Raster object*

Description

You can use the setValues function to assign values to a RasterLayer or RasterBrick object. While you can access the 'values' slot of the objects directly, you would do that at your own peril because when setting values, multiple slots need to be changed; which is what these functions do.

Usage

```
setValues(x, values, layer)
values(x) <- value
```

Arguments

x	A RasterLayer or RasterBrick object
values	Cell values to associate with the RasterLayer object. There should be values for all cells
value	Cell values to associate with the RasterLayer object. There should be values for all cells
layer	Layer number (only relevant for RasterBrick objects). If missing, the values of all layers is set

Value

a Raster* object

Author(s)

Robert J. Hijmans

See Also

replacement

Examples

```
r <- raster(ncol=10, nrow=10)
vals <- 1:ncell(r)
r <- setValues(r, vals)
# equivalent to
r[] <- vals
```

---

shift *Shift*

Description

Shift the location of a Raster* object in the x and/or y direction

Usage

```
shift(object, ...)
```

Arguments

object	A Raster* object
...	Additional arguments, see Details

Details

The following additional arguments can be passed, to replace default values for this function

x	Numeric. The shift in horizontal direction
y	Numeric. The shift in vertical direction
filename	Character. Output filename
format	Character. Output file type. See writeRaster
datatype	Character. Output data type. See dataType
overwrite	Logical. If TRUE, "filename" will be overwritten if it exists
progress	Character. "text", "window", or "" (the default, no progress bar)

Value

a Raster* object

Author(s)

Robert J. Hijmans

See Also

`flip`, `rotate`

Examples

```
r <- raster()
r <- shift(r, x=1, y=-1)
```

`Slope and aspect` *Slope and aspect*

Description

This is a depracated function. Use `terrain` in stead.

Usage

```
slopeAspect(dem, filename='', out=c('slope', 'aspect'), unit='radians', neighbors=8, flatAspect, ...)
```

Arguments

`dem`	RasterLayer object with elevation values in map units, or in meters when the crs is longitude/latitude
`filename`	Character. Filename. optional
`out`	Character vector containing one or more of these options: 'slope', 'aspect'
`unit`	Character. 'degrees' or 'radians'
`neighbors`	Integer. Indicating how many neighboring cells to use to compute slope for any cell. Either 8 (queen case) or 4 (rook case), see Details
`flatAspect`	Numeric or NA. What value to use for aspect when slope is zero (and hence the aspect is undefined)? The default value is 90 degrees (or 0.5*pi radians)
`...`	Standard additional arguments for writing RasterLayer files

Author(s)

Robert J. Hijmans

See Also

`terrain`

spplot — *Use spplot to plot a Raster* object*

Description

A wrapper functions around spplot (sp package). With spplot it is easy to map several layers with a single legend for all maps. ssplot is itself a wrapper around the levelplot function in the lattice package, and see the help for these functions for additional options.

One of the advantages of these wrapper functions is the additional `maxpixels` argument to sample large Raster objects for faster drawing.

Methods

```
spplot(obj, ..., maxpixels=50000, as.table=TRUE)
```

`obj`	A Raster* object
`...`	Any argument that can be passed to `spplot` and levelplot
`maxpixels`	Integer. Number of pixels to sample from each layer of large Raster objects

Author(s)

Robert J. Hijmans

See Also

`plot`, `plotRGB`

The rasterVis package has more advanced plotting methods for Raster objects

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
s <- stack(r, r*2)
layerNames(s) <- c('meuse', 'meuse x 2')

spplot(s)
```

stack — *Create a RasterStack object*

Description

A RasterStack is a collection of RasterLayer objects with the same spatial extent and resolution. A RasterStack can be created from RasterLayer objects, or from raster files, or both. It can also be created from SpatialPixels or SpatialGrid objects.

Value

RasterStack

Methods

`stack(x, ..., bands=NULL)`

filename (character), Raster* object, SpatialGrid*, SpatialPixels*, or list (of filenames and/or Raster* objects). If `x` is a list, additional arguments `...` are ignored.

`..` `x` additional Raster* objects (or filenames)

`bands` Which bands (layers) of the file should be used (default is all layers)

Author(s)

Robert J. Hijmans

See Also

`addLayer`, `dropLayer`, `raster`, `brick`

Examples

```
# file with one layer
fn <- system.file("external/test.grd", package="raster")
s <- stack(fn, fn)
r <- raster(fn)
s <- stack(r, fn)
nlayers(s)

# file with three layers
slogo <- stack(system.file("external/rlogo.grd", package="raster"))
nlayers(slogo)
slogo
```

---

stackApply *Apply a function on subsets of a RasterStack or RasterBrick*

Description

Apply a function on subsets of a RasterStack or RasterBrick. The layers to be combined are indicated with the vector `indices`. The function used should return a single value, and the number of layers in the output Raster* equals the number of unique values in `indices`. For example, if you have a RasterStack with 6 layers, you can use `indices=c(1,1,1,2,2,2)` and `fun=sum`. This will return a RasterBrick with two layers. The first layer is the sum of the first three layers in the input RasterStack, and the second layer is the sum of the last three layers in the input RasterStack. See `calc` if you want to use a function that returns multiple layers based on _all_ layers in the Raster* object.

Usage

```
stackApply(x, indices, fun, filename='', na.rm=TRUE, ...)
```

Arguments

x	A Raster* object
indices	A vector of length nlayers(x) containing all integer values between 1 and the number of layers of the output Raster*
fun	A function that returns a single value, e.g. mean or min, and that takes an 'na.rm' argument
na.rm	Logical. If TRUE, NA cells are removed from calculations
filename	Character. Optional output filename
...	Additional arguments as for writeRaster

Value

A new Raster* object, and in some cases the side effect of a new file on disk.

Author(s)

Robert J. Hijmans

See Also

calc, stackSelect

Examples

```
r <- raster(ncol=10, nrow=10)
r[]=1:ncell(r)
s <- brick(r,r,r,r,r,r)
s <- s * 1:6
b1 <- stackApply(s, indices=c(1,1,1,2,2,2), fun=sum)
b1
b2 <- stackApply(s, indices=c(1,2,3,1,2,3), fun=sum)
b2
```

---

stackSelect *Select cell values from a multi-layer Raster* object*

Description

Use a Raster* object to select cell values from different layers in a multi-layer Raster* object. The object to select values y should have cell values between 1 and nlayers(x). The values of y are rounded.

See extract for extraction of values by cell, point, or otherwise.

Usage

```
## S4 method for signature 'RasterStackBrick,Raster'
stackSelect(x, y, recycle=FALSE, type='index', filename='', ...)
```

Arguments

x	RasterStack or RasterBrick object
y	Raster* object
recycle	Logical. Recursively select values (default = FALSE. Only relevant if y has multiple layers. E.g. if x has 12 layers, and y has 4 layers, the indices of the y layers are used three times.
type	Character. Only relevant when recycle=TRUE. Can be 'index' or 'truefalse'. If it is 'index', the cell values of y should represent layer numbers. If it is 'truefalse' layer numbers are indicated by 0 (not used, NA returned) and 1 (used)
filename	Character. Output filename (optional)
...	Additional arguments as for writeRaster

Value

Raster* object

Author(s)

Robert J. Hijmans

See Also

stackApply, extract

Examples

```
r <- raster(ncol=10, nrow=10)
r[] <- 1
s <- stack(r, r+2, r+5)
r[] <- round((runif(ncell(r)))*3)
x <- stackSelect(s, r)
```

---

subset *Subset layers in a Raster* object*

Description

Extract layers from a Raster* object.

Usage

```
subset(x, ...)
```

Arguments

x	RasterBrick or RasterStack object
...	Additional arguments: subset, which should indicate the layers (represented as integer or by their name). drop=TRUE. If TRUE, a selection of a single layer will be returned as a RasterLayer

Value

Raster* object

Author(s)

Robert J. Hijmans

See Also

dropLayer

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
s <- stack(r,r,r)
sel <- subset(s, 2:3)
sel <- subset(s, 2)
sel <- subset(s, 2, drop=FALSE)

#  sel <- subset(s, 2:3) is equivalent to
sel <- dropLayer(s, 1)
```

---

substitute *Substitute values in a RasterLayer*

Description

Substitute (replace) values in a RasterLayer with values in a data.frame. The data.frame should have a column to identify the key (ID) to match with the values of the RasterLayer, and a column with the replacement values. By default these are the first and second column but you can specify other columns with arguments by and which (see under Methods).

Usage

```
subs(x, y, ...)
```

Arguments

x	A RasterLayer object
y	A data.frame
...	Additional arguments. See under Methods

Details

You could obtain the same result with `reclass`, but `subs` is more efficient for simple replacement. Use reclass if you want to replace ranges of values with new values.

You can also replace values using a fitted model. E.g. fit a model to `glm` or `loess` and then call predict

Value

A new RasterLayer object, and in some cases, the side effect of a new file on disk.

Methods

A full call to `subs` is:

```
subs(x, y, by=1, which=2, subsWithNA=TRUE, filename="", ... )
```

RasterLayer

`y` data.frame

`by` The column number or name that is the key (ID) to match a the `data.frame` row to a value of the RasterLayer. Default is 1

`which` The column number or name that has the new (replacement) values. Default is 2

`subsWithNA` Logical. If `TRUE` values that are not matched become NA. If `FALSE`, they retain their original value (which could also be NA. This latter option is handy when you want to replace only one or a few values).

`filename` Character. output filename

`...` Standard additional variables, see below

`overwrite` Logical. If TRUE, "filename" will be overwritten if it exists

`format` Character. output file type. Either 'raster', 'ascii' or a supported GDAL 'driver' name see `writeFormats`

`datatype` Character. Output data type. See dataType

`progress` Character. "text", "window", or "" (the default, no progress bar)

Author(s)

Robert J. Hijmans

See Also

`reclass`

Examples

```
r <- raster(ncol=10, nrow=10)
r[] <- round(runif(ncell(r)) * 10)
df = data.frame(id=2:8, v=c(10,10,11,11,12:14))
x = subs(r, df)
x2 = subs(r, df, subsWithNA=FALSE)
```

Summary *Summary*

Description

Summarize a Raster* object. A sample is used for very large files.

Details

`summary(x, maxsamp=5000, ...)` You can set the sample size with `maxsamp`

Value

A RasterSummary object

See Also

`cellStats`

Summary-methods *Summary methods*

Description

The following summary methods are available for Raster* objects:

`mean, max, min, range, prod, sum, any, all`

All methods take `na.rm` as an additional logical argument. Default is `na.rm=FALSE`. If `TRUE`, NA values are removed from calculations. These methods compute a summary statistic based on cell values of RasterLayers and the result of these methods is always a single RasterLayer. See `calc` for functions not included here (e.g. median) or any other custom functions.

You can mix RasterLayer, RasterStack and RasterBrick objects with single numeric or logical values. However, because generic functions are used, the method applied is chosen based on the first argument: 'x'. This means that if `r` is a RasterLayer object, `mean(r, 5)` will work, but `mean(5, r)` will not work.

The generic function `range` returns 2 values (the minimum and maximum value of a vector). The Raster* implementations returns a single values (the maximum minus the minimum)

To summarize all cells within a single RasterLayer, see `cellStats` and `maxValue` and `minValue`

Value

a RasterLayer

Author(s)

Robert J. Hijmans

See Also

calc

Examples

```
r1 <- raster(nrow=10, ncol=10)
r1 <- setValues(r1, runif(ncell(r1)))
r2 <- setValues(r1, runif(ncell(r1)))
r3 <- setValues(r1, runif(ncell(r1)))
r <- max(r1, r2, r3)
r <- range(r1, r2, r3, 1.2)

s <- stack(r1, r2, r3)
r <- mean(s, 2)
```

---

terrain *Terrain characteristics*

Description

Compute slope, aspect and other terrain characteristics from a raster with elevation data. The elevation data should be in map units (typically meter) for projected (planar) raster data. They should be in meters when the coordinate reference system (CRS) is longitude/latitude.

This function is the replacement for the depracated function slopeAspect

Usage

```
terrain(x, opt='slope', unit='radians', neighbors=8, filename='', ...)
```

Arguments

x	RasterLayer object with elevation values. Values should have the same unit as the map units, or in meters when the crs is longitude/latitude
opt	Character vector containing one or more of these options: slope, aspect, TPI, TRI, roughness, flowdir (see Details)
unit	Character. 'degrees' or 'radians'. Only relevant for slope and aspect
neighbors	Integer. Indicating how many neighboring cells to use to compute slope for any cell. Either 8 (queen case) or 4 (rook case). Only used for slope and aspect, see Details
filename	Character. Filename. optional
...	Standard additional arguments for writing Raster* objects to file

Details

When `neighbors=4`, slope and aspect are computed according to Fleming and Hoffer (1979) and Ritter (1987). When `neigbors=8`, slope and aspect are computed according to Horn (1981). The Horn algorithm may be best for rough surfaces, and the Fleming and Hoffer algorithm may be better for smoother surfaces (Jones, 1997; Burrough and McDonnell, 1998). If slope = 0, aspect is set to 0.5*pi radians (or 90 degrees if unit='degrees'). When computing slope or aspect, the CRS (`projection`) of the RasterLayer `x` must be known (may not be `NA`), to be able to safely differentiate between planar and longitude/latitude data.

flowdir returns the 'flow direction' (of water), i.e. the direction of the greatest drop in elevation (or the smallest rise if all neighbors are higher). The are enocded as powers of 2 (0 to 8). The cell right to the focal cell 'x' is 1, the one below that is 2, and so on:

64	128	256
32	x	1
16	4	2

If two cells have the same drop in elevation, a random cell is picked. That is not ideal as it may prevent the creation of connected flow networks. ArcGIS implements to the approach of Greenlee (1987) and I might adopt that in the future.

The terrain indices are according to Wilson et al. (2007), as in gdaldem. TRI (Terrain Ruggedness Index) is the mean of the absolute differences between the value of a cell and the value of its 8 surrounding cells. TPI (Topographic Position Index) is the difference between the value of a cell and the mean value of its 8 surrounding cells. Roughness is the difference between the maximum and the minimum value of a cell and its 8 surrounding cells.

Such measures can also be computed with the `focal` function:

```
f <- matrix(1, nrow=3, ncol=3)
TRI <- focal(x, w=f, fun=function(x, ...) sum(abs(x[-5]-x[5]))/8 )
TPI <- focal(x, w=f, fun=function(x, ...) x[5] - mean(x[-5]))
rough <- focal(x, w=f, fun=function(x, ...) max(x) - min(x))
```

Author(s)

Robert J. Hijmans

References

Burrough, P., and R.A. McDonnell, 1998. Principles of Geographical Information Systems. Oxford University Press.

Fleming, M.D. and Hoffer, R.M., 1979. Machine processing of landsat MSS data and DMA topographic data for forest cover type mapping. LARS Technical Report 062879. Laboratory for Applications of Remote Sensing, Purdue University, West Lafayette, Indiana.

Greenlee, D.D., 1987. Raster and vector processing for scanned linework. Photogrammetric Engineering and Remote Sensing 53:1383-1387

Horn, B.K.P., 1981. Hill shading and the reflectance map. Proceedings of the IEEE 69:14-47

Jones, K.H., 1998. A comparison of algorithms used to compute hill slope as a property of the DEM. Computers & Geosciences 24: 315-323

Ritter, P., 1987. A vector-based slope and aspect generation algorithm. Photogrammetric Engineering and Remote Sensing 53: 1109-1111

Wilson, M.F.J., O'Connell, B., Brown, C., Guinan, J.C., Grehan, A.J., 2007. Multiscale terrain analysis of multibeam bathymetry data for habitat mapping on the continental slope. Marine Geodesy 30: 3-35.

See Also

hillShade

Examples

```
## Not run:
elevation <- getData('alt', country='CHE')
x <- terrain(elevation, opt=c('slope', 'aspect'), unit='degrees')
plot(x)

## End(Not run)
```

---

transpose *Transpose*

Description

Transpose a Raster* object

Usage

```
t(x)
```

Arguments

x	a Raster* object

Value

RasterLayer or RasterBrick

Author(s)

Robert J. Hijmans

See Also

transpose: flip, rotate

Examples

```
r <- raster(nrow=18, ncol=36)
r[] <- 1:ncell(r)
rt <- t(r)
```

`trim` *Trim*

Description

Trim (shrink) a Raster* object by removing outer rows and columns that have no data.

Or remove the whitespace before or after a string of characters.

Usage

```
trim(x, ...)
```

Arguments

`x`	A Raster* object or a character string
`...`	additional arguments. See Details.

Details

If `x` is a Raster* object, the following additional arguments can be passed, to replace default values for this function

`padding`	Integer. Number of outer rows/columns with NA values to keep; default=0
`filename`	Output filename. Default = ''
`format`	Character. Output file type. See writeRaster
`datatype`	Character. Output data type; can be 'INT', 'FLT', or a complete datatype description, See dataType
`overwrite`	Logical. If TRUE, "filename" will be overwritten if it exists
`progress`	Character. "text", "window", or "" (the default, no progress bar)

Value

A RasterLayer or RasterBrick object (if `x` is a Raster* object).

A character string If `x` is a character string.

Author(s)

Robert J. Hijmans and Jacob van Etten

Examples

```
r1 <- raster(ncol=36,nrow=18)
```

```
r1[309:310] <- 1:2
r1[353:355] <- 3:5
r1[400] <- 6
s <- trim(r1)


trim("  hi folks  ")
```

---

union *Union Extent*

Description

Union of two Extent objects. See crop and expand to union a Raster object with an Extent object.

Usage

```
## S4 method for signature 'Extent,Extent'
union(x, y)
```

Arguments

x	Extent
y	Extent

Value

Extent

Author(s)

Robert J. Hijmans

See Also

intersect, extent, setExtent

Examples

```
e1 <- extent(-10, 10, -20, 20)
e2 <- extent(0, 20, -40, 5)
union(e1, e2)
```

unionExtent *Extent union*

Description

Obsolete. See union

Usage

```
unionExtent(x, ...)
```

Arguments

x	A Extent or Raster* object
...	additional Extent or Raster* objects

Value

Extent object

Author(s)

Robert J. Hijmans

See Also

union, intersect, extent

Examples

```
e1 <- extent(-10, 10, -20, 20)
e2 <- extent(0, 20, -40, 5)
unionExtent(e1, e2)
```

unique *Unique values*

Description

This function returns the unique values in a RasterLayer

Usage

```
unique(x, incomparables=FALSE, ...)
```

Arguments

x	A RasterLayer object
incomparables	Ignored. Must be missing
...	Ignored. Must be missing

Value

a vector

Author(s)

Robert J.Hijmans

See Also

unique

Examples

```
r <- raster(ncol=10, nrow=10)
r[] <- round(runif(ncell(r))*10)
unique(r)
```

---

unstack *Unstack*

Description

Create or list of RasterLayer objects from a RasterStack or RasterBrick

Usage

```
unstack(x, ...)
```

Arguments

x	a RasterStack object
...	not used. further arguments passed to or from other methods

Value

A list of RasterLayer objects

Author(s)

Robert J. Hijmans

See Also

stack

Examples

```
file <- system.file("external/test.grd", package="raster")
s <- stack(file, file)
list1 <- unstack(s)
b <- brick(s)
list2 <- unstack(b)
```

---

update *Update raster cells of files (on disk)*

Description

Update cell values of a file (i.e., cell values on disk) associated with a RasterLayer or RasterBrick.

User beware: this function _will_ make changes to your file (first make a copy if you are not sure what you are doing).

Writing starts at a cell number `cell`. You can write a vector of values (in cell order), or a matrix. You can also provide a vector of cell numbers (of the same lenght as vector v) to update individual cells.

See writeFormats for supported formats.

Usage

```
update(object, ...)
```

Arguments

object	RasterLayer or RasterBrick that is associated with a file
...	Additional arguments. v - vector or matrix with new values cell - cell from where to start writing. Or a vector of cell numbers if v is a vector of the same lenght. band - band (layer) to update (for RasterBrick objects).

Value

RasterLayer or RasterBrick

Author(s)

Robert J. Hijmans

Examples

```
# setting up an example RasterLayer with file
r <- raster(nrow=5, ncol=10)
r[] = 0
r <- writeRaster(r, 'test', overwrite=TRUE, datatype='INT2S')
as.matrix(r)

# update with a vector starting a cell
r <- update(r, v=rep(1, 5), cell=6)
# 99.99 gets rounded because this is an integer file
r <- update(r, v=9.99, cell=50)
as.matrix(r)

# update with a vector of values and matching vector of cell numbers
r <- update(r, v=5:1, cell=c(5,15,25,35,45))
as.matrix(r)

# updating with a marix, anchored at a cell number
m = matrix(1:10, ncol=2)
r <- update(r, v=m, cell=2)
as.matrix(r)
```

---

validCell *Validity of a cell, column or row number*

Description

Simple helper functions to determine if a row, column or cell number is valid for a certain Raster* object

Usage

```
validCell(object, cell)
validCol(object, colnr)
validRow(object, rownr)
```

Arguments

object	Raster* object (or a SpatialPixels* or SpatialGrid* object)
cell	cell number(s)
colnr	column number; or vector of column numbers
rownr	row number; or vector of row numbers

Value

logical value

Author(s)

Robert J. Hijmans

Examples

```
#using a new default raster (1 degree global)
r <- raster()
validCell(r, c(-1, 0, 1))
validRow(r, c(-1, 1, 100, 10000))
```

---

which *Which cells are TRUE?*

Description

`Which` returns a RasterLayer with `TRUE` or `FALSE`. Cells with 1 are set to `TRUE`, all other cells, including `NA` are set to `FALSE`

Usage

```
Which(x, ...)
```

Arguments

`x`	A RasterLayer object or a logical expression resulting in a RasterLayer object
`...`	Additional arguments. See Details

Details

You can use the additional logical argument `cells`. If cells=`FALSE` (the default), a (logical) RasterLayer is returned, if cells=`TRUE`, the cell numbers are returned for which the condition is `TRUE`

Value

a RasterLayer object

Author(s)

Robert J.Hijmans

See Also

`which`

Examples

```
r <- raster(ncol=10, nrow=10)
r[] <- runif(ncell(r))
r[r< 0.2] <- NA
s <- Which(r > 0.5, cells=TRUE)
s[1:5]
s <- Which(r > 0.5)
```

`writeFormats` *File types for writing*

Description

List supported file types for writing RasterLayer values to disk.

When a function writes a file to disk, the file format is determined by the 'format=' argument if supplied, or else by the file extension (if the extension is known). If other cases the default format is used. The 'factory-fresh' default format is 'raster', but this can be changed using `setOptions`.

Usage

```
writeFormats()
```

Details

writeFormats returns a matrix of the file formats (the "drivers") that are supported.

Supported formats include:

File type	Long name	default extension	Multiband support
raster	'Native' raster package format	.grd	Yes
ascii	ESRI Ascii	.asc	No
SAGA	SAGA GIS	.sdat	No
IDRISI	IDRISI	.rst	No
CDF	netCDF (requires ncdf)	.nc	Yes
GTiff	GeoTiff (requires rgdal)	.tif	Yes
ENVI	ENVI .hdr Labelled	.envi	Yes
EHdr	ESRI .hdr Labelled	.bil	Yes
HFA	Erdas Imagine Images (.img)	.img	Yes

Author(s)

Robert J. Hijmans

See Also

`GDALDriver-class`

Examples

```
writeFormats()
```

`writeRaster` *Write raster data to a file*

Description

Write an entire Raster* object to a file, using one of the many supported formats. See `writeValues` for writing in chunks (e.g. by row).

When writing a file to disk, the file format is determined by the 'format=' argument if supplied, or else by the file extension (if the extension is known). If other cases the default format is used. The 'factory-fresh' default format is 'raster', but this can be changed using `setOptions`.

Usage

```
## S4 method for signature 'RasterLayer,character'
writeRaster(x, filename, format, ...)

## S4 method for signature 'RasterStackBrick,character'
writeRaster(x, filename, format, ...)
```

Arguments

`x`	Raster* object
`filename`	Output filename
`format`	Character. Output file type. See `writeFormats`. If this argument is not provided, it is attempted to infer it from the filename extension. If that fails, the default format is used. The default format is 'raster', but this can be changed using `setOptions`

`...`	Additional arguments:
	`datatype`: Character. Output data type (e.g. 'INT2S' or 'FLT4S'). See `dataType`
	`overwrite`: Logical. If TRUE, "filename" will be overwritten if it exists
	`NAflag`: Numeric. To overwrite the default value used to represent `NA` in a file
	`bandorder`: Character. 'BIL', 'BIP', or 'BSQ'. For 'native' file formats only. For some other formats you can use the 'options' argument (see below)
	`options`: Character. File format specific GDAL options. E.g., when writing a geotiff file: `options=c("COMPRESS=NONE", "TFW=YES")`
	NetCDF files have the following additional, optional, arguments: `varname`, `varunit`, `longname`, `xname`, `yname`, `zname`, `zunit`.

Details

See `writeFormats` for supported file types ("formats", "drivers").

The rgdal package is needed, except for these file formats: 'raster', 'BIL', 'BIP', 'BSQ', 'SAGA', 'ascii', 'IDRISI', and 'CDF'. Some of these formats can be used with or without rgdal (idrisi, SAGA, ascii). You need the 'ncdf' library for the 'CDF' format.

In multi-band files (i.e. files saved from RasterStack or RasterBrick objects), in the native 'raster' format, the bandorder can be set to BIL ('Bands Interleaved by Line'), BIP ('Bands Interleaved by Pixels') or BSQ ('Bands SeQuential'). Note that bandorder is not the same as filetype here.

Supported file types include:

File type	Long name	default extension	Multiband support
`raster`	'Native' raster package format	.grd	Yes
`ascii`	ESRI Ascii	.asc	No
`SAGA`	SAGA GIS	.sdat	No
`IDRISI`	IDRISI	.rst	No
`CDF`	netCDF (requires ncdf)	.nc	Yes
`GTiff`	GeoTiff (requires rgdal)	.tif	Yes
`ENVI`	ENVI .hdr Labelled	.envi	Yes
`EHdr`	ESRI .hdr Labelled	.bil	Yes
`HFA`	Erdas Imagine Images (.img)	.img	Yes

Value

This function is used for the side-effect of writing values to a file.

Author(s)

Robert J. Hijmans

See Also

`writeFormats`, `writeValues`

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))

# take a small part
r <- crop(r, extent(179880, 180800, 329880, 330840) )

# write to an integer binary file
rf <- writeRaster(r, filename="allint.grd", datatype='INT4S', overwrite=TRUE)

# make a brick and save multi-band file
b <- brick(r, sqrt(r))
bf <- writeRaster(b, filename="multi.grd", bandorder='BIL', overwrite=TRUE)
```

```
# write to a new geotiff file (depends on rgdal)
if (require(rgdal)) {
  rf <- writeRaster(r, filename="test.tif", format="GTiff", overwrite=TRUE)
  bf <- writeRaster(b, filename="multi.tif", options="INTERLEAVE=BAND", overwrite=TRUE)
}

# write to netcdf
if (require(ncdf)) {
 rnc <- writeRaster(r, filename='netCDF.nc', format="CDF", overwrite=TRUE)
}
```

`writeValues` *Write values to a file*

Description

Functions for writing blocks (>= 1 row(s)) of values to files. Writing has to start at the first cell of a row (identified with argument `start`) and the values written must represent 1 or more entire rows. Begin by opening a file with `writeStart`, then write values to it in chunks. When writing is done close the file with `writeStop`.

If you want to write all values of a Raster* object at once, you can also use `writeRaster` which is easier to use but more limited. The functions described here allow writing values to file using chunks of different sizes (e.g. 1 or 10 rows). Function `blockSize` can be used to suggest a chunk size to use.

Usage

```
writeStart(x, filename, ...)
writeValues(x, v, start)
writeStop(x)
```

Arguments

`x`	Raster* object
`filename`	Output filename
`...`	Additional arguments as for `writeRaster`
`v`	vector (RasterLayer) or matrix (RasterBrick) of values
`start`	Integer. Row number (counting starts at 1) from where to start writing `v`

Value

RasterLayer or RasterBrick

Author(s)

Robert J. Hijmans

See Also

writeRaster, blockSize, update

Examples

```
r <- raster(system.file("external/test.grd", package="raster"))
# write to a new binary file in chunks
s <- raster(r)
#
tr <- blockSize(r)
tr
s <- writeStart(s, filename='test.grd',  overwrite=TRUE)
for (i in 1:tr$n) {
v <- getValuesBlock(r, row=tr$row[i], nrows=tr$nrows[i])
s <- writeValues(s, v, tr$row[i])
}
s <- writeStop(s)

if(require(rgdal)){
s2 <- writeStart(s, filename='test2.tif', format='GTiff', overwrite=TRUE)
# writing last row first
for (i in tr$n:1) {
v <- getValuesBlock(r, row=tr$row[i], nrows=tr$nrows[i])
s2 <- writeValues(s2, v, tr$row[i])
}
# row number 5 once more
v <- getValuesBlock(r, row=5, nrows=1)
writeValues(s2, v, 5)
s2 <- writeStop(s2)
}

## write values of a RasterStack to a RasterBrick
s <- stack(system.file("external/rlogo.grd", package="raster"))
# create empty brick
b <- brick(s, values=FALSE)
b <- writeStart(b, filename="test.grd", format="raster",overwrite=TRUE)
tr <- blockSize(b)
for (i in 1:tr$n) {
v <- getValuesBlock(s, row=tr$row[i], nrows=tr$nrows[i])
b <- writeValues(b, v, tr$row[i])
}
b <- writeStop(b)
# note that the above is equivalent to
# b <- writeRaster(s, filename="test.grd", format="raster",overwrite=TRUE)
```

---

xyFromCell *Coordinates from a row, column or cell number*

Description

These functions get coordinates of the center of raster cells for a row, column, or cell number of a Raster* object.

Usage

```
xFromCol(object, colnr)
yFromRow(object, rownr)
xyFromCell(object, cell, spatial=FALSE)
xFromCell(object, cell)
yFromCell(object, cell)
```

Arguments

object	Raster* object (or a SpatialPixels* or SpatialGrid* object)
cell	cell number(s)
colnr	column number; or vector of column numbers
rownr	row number; or vector of row numbers
spatial	return a SpatialPoints object (sp package) instead of a matrix

Details

Cell numbers start at 1 in the upper left corner, and increase from left to right, and then from top to bottom The last cell number equals the number of cells of the Raster* object.

Value

xFromCol, yFromCol, xFromCell, yFromCell: vector of x or y coordinate(s) xyFromCell: matrix(x,y) of pairs of coordinates

Author(s)

Robert J. Hijmans

See Also

cellFromXY

Examples

```
#using a new default raster (1 degree global)
r <- raster()
xFromCol(r, c(1, 120, 180))
yFromRow(r, 90)
xyFromCell(r, 10000)
xyFromCell(r, c(0, 1, 32581, ncell(r), ncell(r)+1))

#using a file from disk
r <- raster(system.file("external/test.grd", package="raster"))
```

```
r
cellFromXY(r, c(180000, 330000))
#xy for corners of a raster:
xyFromCell(r, c(1, ncol(r), ncell(r)-ncol(r)+1, ncell(r)))
```

---

z-values *Get or set z-values*

Description

Initial functions for a somewhat more formal approach to get or set z values (e.g. time) associated with layers of Raster* objects. In development.

Usage

```
setZ(x, z, name='time')
getZ(x)
```

Arguments

x	Raster* object
z	vector of z values of any type (e.g. of class 'Date')
name	character label

Value

setZ: Raster* object

getZ: vector

Author(s)

Robert J. Hijmans

Examples

```
r <- raster(ncol=10, nrow=10)
s <- stack(lapply(1:3, function(x) setValues(r, runif(ncell(r)))))
s <- setZ(s, as.Date('2000-1-1') + 0:2)
s
getZ(s)
```

zApply *z (time) apply*

Description

Experimental function to apply a function over a (time) series of layers of a Raster object

Usage

```
zApply(x, by, fun=mean, name='', ...)
```

Arguments

x	Raster* object
by	aggregation indices or function
fun	function to compute aggregated values
name	character label of the new time series
...	additional arguments

Value

Raster* object

Author(s)

Oscar Perpinan Lamigueiro & Robert J. Hijmans

Examples

```
# 12 values of irradiation, 1 for each month
G0dm=c(2.766,3.491,4.494,5.912,6.989,7.742,7.919,7.027,5.369,3.562,2.814,2.179)*1000;
# RasterBrick with 12 layers based on G0dm + noise
r <- raster(nc=10, nr=10)
s <- brick(lapply(1:12, function(x) setValues(r, G0dm[x]+100*rnorm(ncell(r)) )))

# time
tm <- seq(as.Date('2010-01-15'), as.Date('2010-12-15'), 'month')
s <- setZ(s, tm, 'months')

# library(zoo)
# x <- zApply(s, by=as.yearqtr, fun=mean, name='quarters')
```

`zonal` *Zonal statistics*

Description

Compute zonal statistics, values of a Raster* object for each "zone" in a RasterLayer.

Usage

```
zonal(x, zones, stat='mean', digits=0, na.rm=TRUE, progress='')
```

Arguments

`x`	Raster* object
`zones`	RasterLayer object with codes representing zones
`stat`	The function to be applied. Either as character: 'mean', 'min', 'max', 'sum'; or a function (see Details)
`digits`	Integer. Number of digits to maintain in 'zones'. By default averaged to an integer (zero digits)
`na.rm`	Logical. If `TRUE`, `NA` values in `x` are ignored
`progress`	Character. "text", "window", or "" (the default, no progress bar)

Details

If `stat` is a `function`, `zonal` will fail (gracefully) for very large RasterLayers. The function should accept a `na.rm` argument. For example, `fun=length` will fail, but `fun=function(x, ...){length(x)}` will work; the '...' arguments catches the na.rm argument, even though it is not used by the function in this case. To remove `NA` values, you could use this function: `fun=function(x, na.rm){ if(na.rm){length(na.omit(x))}else{length(x)}}`

Value

A matrix with a value for each zone (unique value in `zones`)

Author(s)

Robert J. Hijmans

See Also

See `cellStats` for 'global' statistics (i.e., all of `x` is considered a single zone)

Examples

```
r <- raster(ncols=10, nrows=10)
r[] <- runif(ncell(r)) * 1:ncell(r)
z <- r
z[] <- rep(1:5, each=20)
zonal(r, z, mean)
zonal(r, z, min)
# for big files, use a character value rather than a function
zonal(r, z, 'sum')

# multiple layers
zonal(stack(r, r*10), z, 'sum')
```

---

zoom *Zoom in on a plot*

Description

Zoom in on a plot (map) by providing a new extent, by default this is done by clicking twice on the map

Usage

```
zoom(x, ...)
## S4 method for signature 'Raster'
zoom(x, ext=drawExtent(), maxpixels=100000, layer=1, new=TRUE, useRaster=TRUE, ...)

## S4 method for signature 'Spatial'
zoom(x, ext=drawExtent(), new=TRUE, ...)
```

Arguments

x	Raster* or Spatial* (vector type) object
ext	Extent object
maxpixels	Maximum number of pixels used for the map
layer	Positive integer to select the layer to be used if x is a mutilayer Raster object
new	Logical. If TRUE, the zoomed in map will appear on a new device (window)
useRaster	Logical. If TRUE, a bitmap raster is used to plot the image instead of polygons
...	additional paramters for plot

Value

Extent object (invisibly)

Author(s)

Robert J. Hijmans

See Also

drawExtent, plot

Index