

Package ‘landscapetools’

July 9, 2026

Type Package

Title Landscape Utility Toolbox

Version 0.6.3

Description Provides utility functions for some of the less-glamorous tasks involved in landscape analysis. It includes functions to coerce raster data to the common 'tibble' format and vice versa, it helps with flexible reclassification tasks of raster data and it provides a function to merge multiple raster. Furthermore, 'landscapetools' helps landscape scientists to visualize their data by providing optional themes and utility functions to plot single landscapes, 'rasterstacks', '-bricks' and lists of raster.

License GPL-3

Encoding UTF-8

LazyData true

ByteCompile true

Depends R (>= 3.5)

URL <https://docs.ropensci.org/landscapetools/>

BugReports <https://github.com/ropensci/landscapetools/issues>

Imports ggplot2, raster, tibble, Rcpp, rlang

Suggests testthat, covr, knitr, rmarkdown

VignetteBuilder knitr

LinkingTo Rcpp

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Marco Sciaini [aut] (ORCID: <<https://orcid.org/0000-0002-3042-5435>>),
Matthias Fritsch [aut],
Maximilian H.K. Hesselbarth [aut] (ORCID:
<<https://orcid.org/0000-0003-1125-9918>>),
Jakub Nowosad [aut] (ORCID: <<https://orcid.org/0000-0002-1057-3721>>),
Anatoly Tsyplenkov [aut, cre] (ORCID:
<<https://orcid.org/0000-0003-4144-8402>>),

Laura Graham [rev] (Laura reviewed the package for rOpenSci, see <https://github.com/ropensci/onboarding/issues/188>),
 Jeffrey Hollister [rev] (Jeffrey reviewed the package for rOpenSci, see <https://github.com/ropensci/onboarding/issues/188>)

Maintainer Anatoly Tsyplenkov <s2@sent.com>

Repository CRAN

Date/Publication 2026-07-09 09:10:08 UTC

Contents

classified_landscape	2
fractal_landscape	3
gradient_landscape	3
random_landscape	4
show_landscape	4
show_shareplot	6
theme_nlm	8
util_as_integer	14
util_binarize	14
util_classify	15
util_extract_multibuffer	17
util_merge	18
util_raster2tibble	19
util_rescale	20
util_tibble2raster	20
util_writeESRI	21

Index **23**

classified_landscape *Example map (factor).*

Description

An example map to show landscapetools functionality generated with the nlm_random() algorithm with factorial values.

Usage

```
classified_landscape
```

Format

A raster layer object.

Source

Simulated neutral landscape models with R. <https://github.com/ropensci/NLMR/>

fractal_landscape *Example map (fractional brownian motion).*

Description

An example map to show landscapetools functionality generated with the nlm_fbm() algorithm.

Usage

```
fractal_landscape
```

Format

A raster layer object.

Source

Simulated neutral landscape models with R. <https://github.com/ropensci/NLMR/>

gradient_landscape *Example map (planar gradient).*

Description

An example map to show landscapetools functionality generated with the nlm_planargradient() algorithm.

Usage

```
gradient_landscape
```

Format

A raster layer object.

Source

Simulated neutral landscape models with R. <https://github.com/ropensci/NLMR/>

random_landscape	<i>Example map (random).</i>
------------------	------------------------------

Description

An example map to show landscapetools functionality generated with the nlm_random() algorithm.

Usage

```
random_landscape
```

Format

A raster layer object.

Source

Simulated neutral landscape models with R. <https://github.com/ropensci/NLMR/>

show_landscape	<i>show_landscape</i>
----------------	-----------------------

Description

Plot a Raster* object with the NLMR default theme (as ggplot).

Usage

```
show_landscape(x, xlab, ylab, discrete, unique_scales, n_col, n_row, ...)

## S3 method for class 'RasterLayer'
show_landscape(x, xlab = "Easting", ylab = "Northing", discrete = FALSE, ...)

## S3 method for class 'list'
show_landscape(
  x,
  xlab = "Easting",
  ylab = "Northing",
  discrete = FALSE,
  unique_scales = FALSE,
  n_col = NULL,
  n_row = NULL,
  ...
)

## S3 method for class 'RasterStack'
```

```

show_landscape(
  x,
  xlab = "Easting",
  ylab = "Northing",
  discrete = FALSE,
  unique_scales = FALSE,
  n_col = NULL,
  n_row = NULL,
  ...
)

## S3 method for class 'RasterBrick'
show_landscape(
  x,
  xlab = "Easting",
  ylab = "Northing",
  discrete = FALSE,
  unique_scales = FALSE,
  n_col = NULL,
  n_row = NULL,
  ...
)

```

Arguments

x	Raster* object
xlab	x axis label, default "Easting"
ylab	y axis label, default "Northing"
discrete	If TRUE, the function plots a raster with a discrete legend.
unique_scales	If TRUE and multiple raster are to be visualized, each facet can have a unique color scale for its fill
n_col	If multiple rasters are to be visualized, n_col controls the number of columns for the facet
n_row	If multiple rasters are to be visualized, n_row controls the number of rows for the facet
...	Arguments for theme_nlm

Value

ggplot2 Object

Examples

```

x <- gradient_landscape

# classify
y <- util_classify(gradient_landscape,

```

```

n = 3,
level_names = c("Land Use 1", "Land Use 2", "Land Use 3"))

show_landscape(x)
show_landscape(y, discrete = TRUE)

show_landscape(list(gradient_landscape, random_landscape))
show_landscape(raster::stack(gradient_landscape, random_landscape))

show_landscape(list(gradient_landscape, y), unique_scales = TRUE)

```

show_shareplot	<i>show_shareplot</i>
----------------	-----------------------

Description

Plot the landscape share in subsequential buffers around a/multiple point(s) of interest

Usage

```

show_shareplot(
  landscape,
  points,
  buffer_width,
  max_width = NULL,
  multibuffer_df = NULL,
  return_df = FALSE
)

```

```

show_shareplot(
  landscape,
  points,
  buffer_width,
  max_width = NULL,
  multibuffer_df = NULL,
  return_df = FALSE
)

```

Arguments

landscape	Raster* object
points	Point(s) represented by a two-column matrix or data.frame; SpatialPoints*; SpatialPolygons*; SpatialLines; Extent; a numeric vector representing cell numbers; or sf* POINT object

buffer_width	Buffer widths in which landscape share is measured. By default, it is a vector of buffer sizes, if max_width = NULL. If a value is provided for max_width, a series of buffer sizes is created, from buffer_width to max_width, with increases of buffer_width.
max_width	Max distance to which buffer_width is summed up; the x axis in the plot
multibuffer_df	data.frame with landscape share or a function from it already extracted, such as through the <code>util_extract_multibuffer()</code> function. If given, the other arguments (landscape, points, buffer_width, max_width) are ignored. Default is NULL.
return_df	Logical value indicating if a tibble with the underlying data should be returned

Value

ggplot2 Object

Examples

```
# Minimal runnable example with a pre-built multi-buffer data frame
df <- data.frame(
  id = "Point ID: 1",
  layer = factor(rep(1:3, each = 2)),
  freq = c(10, 15, 20, 25, 5, 10),
  buffer = rep(c(10, 20), 3)
)
show_shareplot(multibuffer_df = df)

# use a smaller aggregated landscape for the longer-running examples below
small_landscape <- raster::aggregate(classified_landscape, fact = 5)

# create single point
new_point <- matrix(c(75, 75), ncol = 2)

# show landscape and point of interest
show_landscape(small_landscape, discrete = TRUE) +
ggplot2::geom_point(data = data.frame(x = new_point[, 1], y = new_point[, 2]),
  ggplot2::aes(x = x, y = y),
  col = "grey", size = 3)

# show single point share
show_shareplot(small_landscape, new_point, 10, 30)

# show multiple points share
new_points <- matrix(c(75, 110, 75, 30), ncol = 2)
show_shareplot(small_landscape, new_points, 10, 30)

# irregular buffer widths
show_shareplot(small_landscape, new_points, c(10, 30))

# get data frame with results back
result <- show_shareplot(small_landscape, new_points, 10, 30, return_df = TRUE)
```

```
result$share_df

# use the output from util_extract_multibuffer
df <- util_extract_multibuffer(small_landscape, new_points, 10, 30)
show_shareplot(multibuffer_df = df)
```

theme_nlm

theme_nlm

Description

Opinionated ggplot2 theme to visualize NLM raster.

Usage

```
theme_nlm(
  base_family = NULL,
  base_size = 11.5,
  plot_title_family = base_family,
  plot_title_size = 18,
  plot_title_face = "bold",
  plot_title_margin = 10,
  subtitle_family = NULL,
  subtitle_size = 13,
  subtitle_face = "plain",
  subtitle_margin = 15,
  strip_text_family = base_family,
  strip_text_size = 12,
  strip_text_face = "plain",
  strip.background = "grey80",
  caption_family = NULL,
  caption_size = 9,
  caption_face = "plain",
  caption_margin = 10,
  axis_text_size = base_size,
  axis_title_family = base_family,
  axis_title_size = 9,
  axis_title_face = "plain",
  axis_title_just = "rt",
  plot_margin = ggplot2::unit(c(0, 0, 0, 0), "lines"),
  grid_col = "#cccccc",
  grid = TRUE,
  axis_col = "#cccccc",
  axis = FALSE,
  ticks = FALSE,
  legend_title = "Z",
```

```
    legend_labels = NULL,  
    legend_text_size = 8,  
    legend_title_size = 10,  
    ratio = 1,  
    viridis_scale = "D",  
    ...  
)  
  
theme_nlm_discrete(  
  base_family = NULL,  
  base_size = 11.5,  
  plot_title_family = base_family,  
  plot_title_size = 18,  
  plot_title_face = "bold",  
  plot_title_margin = 10,  
  subtitle_family = NULL,  
  subtitle_size = 13,  
  subtitle_face = "plain",  
  subtitle_margin = 15,  
  strip_text_family = base_family,  
  strip_text_size = 12,  
  strip_text_face = "plain",  
  strip.background = "grey80",  
  caption_family = NULL,  
  caption_size = 9,  
  caption_face = "plain",  
  caption_margin = 10,  
  axis_text_size = base_size,  
  axis_title_family = base_family,  
  axis_title_size = 9,  
  axis_title_face = "plain",  
  axis_title_just = "rt",  
  plot_margin = ggplot2::unit(c(0, 0, 0, 0), "lines"),  
  grid_col = "#cccccc",  
  grid = TRUE,  
  axis_col = "#cccccc",  
  axis = FALSE,  
  ticks = FALSE,  
  legend_title = "Z",  
  legend_labels = NULL,  
  legend_text_size = 8,  
  legend_title_size = 10,  
  ratio = 1,  
  viridis_scale = "D",  
  ...  
)  
  
theme_nlm_grey(  

```

```
base_family = NULL,  
base_size = 11.5,  
plot_title_family = base_family,  
plot_title_size = 18,  
plot_title_face = "bold",  
plot_title_margin = 10,  
subtitle_family = NULL,  
subtitle_size = 13,  
subtitle_face = "plain",  
subtitle_margin = 15,  
strip_text_family = base_family,  
strip_text_size = 12,  
strip_text_face = "plain",  
strip.background = "grey80",  
caption_family = NULL,  
caption_size = 9,  
caption_face = "plain",  
caption_margin = 10,  
axis_text_size = base_size,  
axis_title_family = base_family,  
axis_title_size = 9,  
axis_title_face = "plain",  
axis_title_just = "rt",  
plot_margin = ggplot2::unit(c(0, 0, 0, 0), "lines"),  
grid_col = "#cccccc",  
grid = TRUE,  
axis_col = "#cccccc",  
axis = FALSE,  
ticks = FALSE,  
legend_title = "Z",  
legend_labels = NULL,  
legend_text_size = 8,  
legend_title_size = 10,  
ratio = 1,  
...  
)
```

```
theme_nlm_grey_discrete(  
  base_family = NULL,  
  base_size = 11.5,  
  plot_title_family = base_family,  
  plot_title_size = 18,  
  plot_title_face = "bold",  
  plot_title_margin = 10,  
  subtitle_family = NULL,  
  subtitle_size = 13,  
  subtitle_face = "plain",  
  subtitle_margin = 15,
```

```
strip_text_family = base_family,
strip_text_size = 12,
strip_text_face = "plain",
strip.background = "grey80",
caption_family = NULL,
caption_size = 9,
caption_face = "plain",
caption_margin = 10,
axis_text_size = base_size,
axis_title_family = base_family,
axis_title_size = 9,
axis_title_face = "plain",
axis_title_just = "rt",
plot_margin = ggplot2::unit(c(0, 0, 0, 0), "lines"),
grid_col = "#cccccc",
grid = TRUE,
axis_col = "#cccccc",
axis = FALSE,
ticks = FALSE,
legend_title = "Z",
legend_labels = NULL,
legend_text_size = 8,
legend_title_size = 10,
ratio = 1,
...
)

theme_facetplot(
  base_family = NULL,
  base_size = 11.5,
  plot_title_family = base_family,
  plot_title_size = 18,
  plot_title_face = "bold",
  plot_title_margin = 10,
  subtitle_family = NULL,
  subtitle_size = 13,
  subtitle_face = "plain",
  subtitle_margin = 15,
  strip.background = "grey80",
  caption_family = NULL,
  caption_size = 9,
  caption_face = "plain",
  caption_margin = 10,
  ratio = 1,
  viridis_scale = "D",
  ...
)
```

```

theme_facetplot_discrete(
  base_family = NULL,
  base_size = 11.5,
  plot_title_family = base_family,
  plot_title_size = 18,
  plot_title_face = "bold",
  plot_title_margin = 10,
  subtitle_family = NULL,
  subtitle_size = 13,
  subtitle_face = "plain",
  subtitle_margin = 15,
  strip.background = "grey80",
  caption_family = NULL,
  caption_size = 9,
  caption_face = "plain",
  caption_margin = 10,
  ratio = 1,
  viridis_scale = "D",
  ...
)

```

Arguments

<code>base_family</code>	base font family size
<code>base_size</code>	base font size
<code>plot_title_family</code>	plot title family
<code>plot_title_size</code>	plot title size
<code>plot_title_face</code>	plot title face
<code>plot_title_margin</code>	plot title <code>ggplot2::margin</code>
<code>subtitle_family</code>	plot subtitle family
<code>subtitle_size</code>	plot subtitle size
<code>subtitle_face</code>	plot subtitle face
<code>subtitle_margin</code>	plot subtitle <code>ggplot2::margin</code> bottom (single numeric value)
<code>strip_text_family</code>	facet facet label font family
<code>strip_text_size</code>	facet label font family, face and size
<code>strip_text_face</code>	facet facet label font face

strip.background	strip background
caption_family	plot caption family
caption_size	plot caption size
caption_face	plot caption face
caption_margin	plot caption ggplot2::margin
axis_text_size	axis text size
axis_title_family	axis title family
axis_title_size	axis title size
axis_title_face	axis title face
axis_title_just	axis title justification
plot_margin	plot ggplot2::margin (specify with 'ggplot2::margin')
grid_col	grid color
grid	grid TRUE/FALSE
axis_col	axis color
axis	axis TRUE/FALSE
ticks	ticks TRUE/FALSE
legend_title	Title of the legend (default "Z")
legend_labels	Labels for the legend ticks, if used with <code>show_landscape</code> they are automatically derived.
legend_text_size	legend text size, default 8
legend_title_size	legend text size, default 10
ratio	ratio for tiles (default 1, if your raster is not a square the ratio should be <code>raster::nrow(x) / raster::ncol(x)</code>)
viridis_scale	Five options are available: "viridis - magma" (= "A"), "viridis - inferno" (= "B"), "viridis - plasma" (= "C"), "viridis - viridis" (= "D", the default option), "viridis - cividis" (= "E")
...	optional arguments to <code>ggplot2::theme</code>

Details

A focused theme to visualize raster data that sets a lot of defaults for the `ggplot2::theme`.

The functions are setup in such a way that you can customize your own one by just wrapping the call and changing the parameters. The theme itself is heavily influenced by `hrbrmstr` and his package `hrbrthemes` (<https://github.com/hrbrmstr/hrbrthemes/>).

Value

A list of `ggplot2` components. The list contains a `ggplot2` theme object and a fill scale object that can be added to a `ggplot` with `+` to style raster visualizations.

util_as_integer	<i>util_as_integer</i>
-----------------	------------------------

Description

Coerces raster values to integers

Usage

```
util_as_integer(x)

## S3 method for class 'RasterLayer'
util_as_integer(x)
```

Arguments

x	raster
---	--------

Details

Coerces raster values to integers, which is sometimes needed if you want further methods that rely on integer values.

Value

RasterLayer

Examples

```
# Mode 1
util_as_integer(fractal_landscape)
```

util_binarize	<i>Binarize continuous raster values</i>
---------------	--

Description

Classify continuous raster values into binary map cells based upon given break(s).

Usage

```
util_binarize(x, breaks)

## S3 method for class 'RasterLayer'
util_binarize(x, breaks)
```

Arguments

x	Raster* object
breaks	Vector with one or more break percentages

Details

Breaks are considered to be habitat percentages (ρ). If more than one percentage is given multiple layers are written in the same brick.

Value

RasterLayer / RasterBrick

Examples

```
breaks <- c(0.3, 0.5)
binary_maps <- util_binarize(gradient_landscape, breaks)
```

util_classify	<i>util_classify</i>
---------------	----------------------

Description

Classify continuous landscapes into landscapes with discrete classes

Usage

```
util_classify(x, n, weighting, level_names, real_land, mask_val)
```

```
## S3 method for class 'RasterLayer'
util_classify(
  x,
  n = NULL,
  weighting = NULL,
  level_names = NULL,
  real_land = NULL,
  mask_val = NULL
)
```

Arguments

x	raster
n	Number of classes
weighting	Vector of numeric values that are considered to be habitat percentages (see details)

level_names	Vector of names for the factor levels.
real_land	Raster with real landscape (see details)
mask_val	Value to mask (refers to real_land)

Details

Mode 1: Calculate the optimum breakpoints using Jenks natural breaks optimization, the number of classes is determined with n. The Jenks optimization seeks to minimize the variance within categories, while maximizing the variance between categories.

Mode 2: The number of elements in the weighting vector determines the number of classes in the resulting matrix. The classes start with the value 1. If non-numerical levels are required, the user can specify a vector to turn the numerical factors into other data types, for example into character strings (i.e. class labels). If the numerical vector of weightings does not sum up to 1, the sum of the weightings is divided by the number of elements in the weightings vector and this is then used for the classificat#' .

Mode 3: For a given 'real' landscape the number of classes and the weightings are extracted and used to classify the given landscape (any given weighting parameter is overwritten in this case!). If an optional mask value is given the corresponding class from the 'real' landscape is cut from the landscape beforehand.

Value

RasterLayer

Examples

```
# Mode 1
util_classify(fractal_landscape,
              n = 3,
              level_names = c("Land Use 1", "Land Use 2", "Land Use 3"))

# Mode 2
util_classify(fractal_landscape,
              weighting = c(0.5, 0.25, 0.25),
              level_names = c("Land Use 1", "Land Use 2", "Land Use 3"))

# Mode 3
real_land <- util_classify(gradient_landscape,
                          n = 3,
                          level_names = c("Land Use 1", "Land Use 2", "Land Use 3"))

fractal_landscape_real <- util_classify(fractal_landscape, real_land = real_land)
fractal_landscape_mask <- util_classify(fractal_landscape, real_land = real_land, mask_val = 1)

landscapes <- list(
  '1 nlm' = fractal_landscape,
  '2 real' = real_land,
  '3 result' = fractal_landscape_real,
  '4 result with mask' = fractal_landscape_mask
)
```

```
show_landscape(landscapes, unique_scales = TRUE, nrow = 1)
```

```
util_extract_multibuffer
```

Extract raster values for multiple buffers

Description

This function creates a series of circular buffers around spatial points and computes the frequency of each value of a raster within the buffers; the results are printed in a data.frame.

Usage

```
util_extract_multibuffer(
  landscape,
  points,
  buffer_width,
  max_width = NULL,
  rel_freq = FALSE,
  fun = NULL,
  point_id_text = TRUE,
  ...
)
```

Arguments

landscape	Raster* object
points	Point(s) represented by a two-column matrix or data.frame; SpatialPoints*; SpatialPolygons*; SpatialLines; Extent; a numeric vector representing cell numbers; or sf* POINT object.
buffer_width	Buffer widths in which the frequency of landscape values is measured. It might be either a single value or a vector of buffer sizes, if max_width = NULL (default). If a value is provided for max_width, a series of buffer sizes is created, from buffer_width to max_width, with increases of buffer_width.
max_width	Maximum distance to which buffer_width is summed up. If NULL, buffer_width is interpreted as a series of buffer widths.
rel_freq	Logical. If TRUE, the relative frequency of raster values is also returned, besides the absolute frequency. Ignored if fun is provided.
fun	Function to apply to raster values within the buffer (e.g. "median", "mean").
point_id_text	Logical. If TRUE, the string "Point ID:" is added to the first column of the output.
...	additional arguments (none implemented)

Value

A tibble with the frequency of each raster value within the buffers of different sizes around each point. Alternatively, a tibble with the relative frequency of raster values, if `rel_freq = TRUE`, or a function from the raster values, if `fun` is provided.

Examples

```
# create single point
new_point <- matrix(c(75,75), ncol = 2)

# show landscape and point of interest
show_landscape(classified_landscape, discrete = TRUE) +
  ggplot2::geom_point(data = data.frame(x = new_point[,1], y = new_point[,2]),
    ggplot2::aes(x = x, y = y),
    col = "grey", size = 3)

# extract frequency of each pixel value within each buffer from 10 to 50 m width
util_extract_multibuffer(classified_landscape, new_point, 10, 50)
# use irregular buffer sizes
util_extract_multibuffer(classified_landscape, new_point, c(5, 10, 20, 30))
# also returns relative frequency
util_extract_multibuffer(classified_landscape, new_point, 10, 50, rel_freq = TRUE)
# use a given function - e.g. median in each buffer width
util_extract_multibuffer(classified_landscape, new_point, 10, 50, fun = "median")

# show multiple points share
new_points <- matrix(c(75, 110, 75, 30), ncol = 2)
util_extract_multibuffer(classified_landscape, new_points, c(5, 10, 20, 30))
```

 util_merge

util_merge

Description

Merge a primary raster with other rasters weighted by scaling factors.

Usage

```
util_merge(primary_nlm, secondary_nlm, scalingfactor = 1, rescale)

## S3 method for class 'RasterLayer'
util_merge(primary_nlm, secondary_nlm, scalingfactor = 1, rescale = TRUE)
```

Arguments

```
primary_nlm    Primary Raster* object
secondary_nlm  A list or stack of Raster* objects that are merged with the primary Raster*
               object
```

scalingfactor Weight for the secondary Raster* objects
rescale If TRUE (default), the values are rescaled between 0-1.

Value

Rectangular matrix with values ranging from 0-1

Examples

```
x <- util_merge(gradient_landscape, random_landscape)
show_landscape(x)
```

util_raster2tibble *Converts raster data into tibble*

Description

Writes spatial raster values into tibble and adds coordinates.

Usage

```
util_raster2tibble(x, format = "long")
```

```
util_raster2tibble(x, format = "long")
```

Arguments

x Raster* object
format Either "long" (default) or "wide" output for the resulting tibble

Details

You will loose any resolution, extent or reference system. The output is raw tiles.

Value

a tibble

Examples

```
maptib <- util_raster2tibble(fractal_landscape)

library(ggplot2)
ggplot(maptib, aes(x,y)) +
  coord_fixed() +
  geom_raster(aes(fill = z))
```

util_rescale	<i>util_rescale</i>
--------------	---------------------

Description

Linearly rescale element values in a raster to a range between 0 and 1.

Usage

```
util_rescale(x)
```

```
util_rescale(x)
```

Arguments

x Raster* object

Details

Rasters generated by nlm_ functions are scaled between 0 and 1 as default, this option can be set to FALSE if needed.

Value

Raster* object with values ranging from 0-1

Examples

```
unscaled_landscape <- gradient_landscape + fractal_landscape  
util_rescale(unscaled_landscape)
```

util_tibble2raster	<i>Converts tibble data into a raster</i>
--------------------	---

Description

Writes spatial tibble values into a raster.

Usage

```
util_tibble2raster(x)
```

```
util_tibble2raster(x)
```

Arguments

x a tibble

Details

Writes tiles with coordinates from a tibble into a raster. Resolution is set to 1 and the extent will be $c(0, \max(x), 0, \max(y))$.

You can directly convert back the result from 'util_raster2tibble()' without problems. If you have altered the coordinates or otherwise played with the data, be careful while using this function.

Value

Raster* object

Examples

```
maptib <- util_raster2tibble(random_landscape)
mapras <- util_tibble2raster(maptib)
all.equal(random_landscape, mapras)
```

util_writeESRI

util_writeESRI

Description

Export raster objects as ESRI ascii files.

Usage

```
util_writeESRI(x, filepath)

## S3 method for class 'RasterLayer'
util_writeESRI(x, filepath)
```

Arguments

x Raster* object
 filepath path where to write the raster to file

Details

raster::writeRaster or SDMTools::write.asc both export files that are recognised by most GIS software, nevertheless they both have UNIX linebreaks. Some proprietary software (like SPIP for example) require an exact 1:1 replica of the output of ESRI's ArcMap, which as a Windows software has no carriage returns at the end of each line. util_writeESRI should therefore only be used if you need this, otherwise raster::writeRaster is the better fit for exporting raster data in R.

Value

No return value, called for the side effect of writing an ESRI ASCII raster file to `filepath`.

Examples

```
filepath <- tempfile(fileext = ".asc")
util_writeESRI(gradient_landscape, filepath)
unlink(filepath)
```

Index

* datasets

- classified_landscape, 2
- fractal_landscape, 3
- gradient_landscape, 3
- random_landscape, 4

classified_landscape, 2

fractal_landscape, 3

gradient_landscape, 3

random_landscape, 4

show_landscape, 4, 13

show_shareplot, 6

theme_facetplot (theme_nlm), 8

theme_facetplot_discrete (theme_nlm), 8

theme_nlm, 5, 8

theme_nlm_discrete (theme_nlm), 8

theme_nlm_grey (theme_nlm), 8

theme_nlm_grey_discrete (theme_nlm), 8

util_as_integer, 14

util_binarize, 14

util_classify, 15

util_extract_multibuffer, 17

util_extract_multibuffer(), 7

util_merge, 18

util_raster2tibble, 19

util_rescale, 20

util_tibble2raster, 20

util_writeESRI, 21