Package 'geometr'

March 30, 2021

Title Generate and Modify Interoperable Geometric Shapes

Version 0.2.8

Description Provides tools that generate and process fully accessible and tidy geometric shapes. The package improves interoperability of spatial and other geometric classes by providing getters and setters that produce identical output from various classes.

URL https://ehrmanns.github.io/geometr/

BugReports https://github.com/EhrmannS/geometr/issues

Depends R (>= 2.10)

Language en-gb

License GPL-3

Encoding UTF-8

LazyData true

Imports checkmate, crayon, deldir, dplyr, grDevices, grid, methods, raster, readr, rgdal, rlang, sf, sp, spatstat, tibble, Rcpp

Suggests testthat, rmarkdown, bookdown, magrittr, covr, knitr

LinkingTo Rcpp

RoxygenNote 7.1.1

SystemRequirements C++11

VignetteBuilder knitr

NeedsCompilation yes

Author Steffen Ehrmann [aut, cre] (<https://orcid.org/0000-0002-2958-0796>), Dan Sunday [cph] (fast point-in-polygon algorithm.)

Maintainer Steffen Ehrmann <steffen.science@funroll-loops.de>

Repository CRAN

Date/Publication 2021-03-30 11:10:06 UTC

R topics documented:

`.getDecimals` *Get the number of decimal places*

Description

Get the number of decimal places

Usage

```
.getDecimals(x)
```

Arguments

`x`	`[numeric(1)]` the number for which to derive decimal places.

.makeGrob *Make the grob of a plot*

Description

Make the grob of a plot

Usage

```
.makeGrob(x, plotParams, theme = gtTheme)
```

Arguments

x	the object to transform to class grob.
plotParams	[named list(.)] new plotting parameters specified via the quick options in visualise.
theme	[gtTheme(1)] the theme from which to take parameters.

Value

Depending on the provided geometry either a pointsGrob, a polylineGrob, a pathGrob or a rasterGrob.

.makeLayout *Make the layout of a plot*

Description

Make the layout of a plot

Usage

```
.makeLayout(legend, window, theme = gtTheme)
```

Arguments

legend	[list(.)] the legend object built with .makeLegend.
window	[data.frame(1)] two opposing corners of a rectangle to which the plot is limited.
theme	[gtTheme(1)] the theme from which to take graphical parameters.

.makeLegend *Make the legend of a plot*

Description

Make the legend of a plot

Usage

```
.makeLegend(x, plotParams, theme)
```

Arguments

x	[list(1)] any spatial object to plot.
plotParams	[named list(.)] new plotting parameters specified via the quick options in visualise.
theme	[list(7)] the theme from which to take graphical parameters.

.makePlot *Make the object to a plot*

Description

Make the object to a plot

Usage

```
.makePlot(x, window, theme = gtTheme, ...)
```

Arguments

x	[list(1)] named list of the object from which to make the plot.
window	[data.frame(1)] two opposing corners of a rectangle to which the plot is limited.
theme	[gtTheme(1)] the theme from which to take graphical parameters.
...	instead of providing a gtTheme, you can also determine specific graphic parameters (see gpar) separately; see setTheme for details.

.makeTinyMap *Make a tiny map*

Description

A tiny map is used via the show method of a geom.

Usage

```
.makeTinyMap(geom = NULL)
```

Arguments

geom	[geom] the geom from which to create a tiny map.

.rad *Convert degree to radians*

Description

Convert degree to radians

Usage

```
.rad(degree)
```

Arguments

degree	[numeric(1)] a degree value to convert to radians.

.testAnchor *Test anchor for consistency*

Description

Test anchor for consistency

Usage

```
.testAnchor(x, ...)
```

Arguments

x	[data.frame | geom] the object to be tested for consistency.
...	[.] additional arguments.

.testPoints *Test points to not contain NA*

Description

Test points to not contain NA

Usage

```
.testPoints(x, ...)
```

Arguments

x	[data.frame] the points to be tested.
...	[.] additional arguments.

.testTemplate *Test template for consistency*

Description

Test template for consistency

Usage

```
.testTemplate(x, ...)
```

Arguments

x	[RasterLayer | matrix] the object to be tested for consistency.
...	[.] additional arguments.

.testWindow *Test window for consistency*

Description

Test window for consistency

Usage

```
.testWindow(x, ...)
```

Arguments

x	[data.frame] the object to be tested for consistency.
...	[.] additional arguments.

.updateOrder *Update column order*

Description

Set the order of the table columns to `c("fid","gid",rest)`

Usage

```
.updateOrder(input = NULL)
```

Arguments

`input`	[`data.frame(1)`] a table that contains at least the columns `fid` and `gid`.

Value

A new table where the columns have the correct order.

.updateTheme *Update the theme parameters*

Description

Update the theme parameters

Usage

```
.updateTheme(x, theme = gtTheme)
```

Arguments

`x`	[`list(1)`] any spatial object to plot.
`theme`	[`gtTheme(1)`] the theme that shall be updated.

.updateVertices *Update the vertices*

Description

Set the vertices in a table so that they are valid for a geom.

Usage

```
.updateVertices(input = NULL)
```

Arguments

input	[data.frame(1)] a table of vertices which should be brought into the correct form.

.updateWindow *Update the window*

Description

Set a window to the minimum/maximum values of input vertices.

Usage

```
.updateWindow(input = NULL, window = NULL)
```

Arguments

input	[data.frame(1)] a table of vertices for which a new window should be derived.
window	[data.frame(1)] the old window.

Value

A new window that has the extent of input.

gc_geom *Transform a spatial object to class* geom

Description

Transform a spatial object to class geom

Usage

```
## S4 method for signature 'Spatial'
gc_geom(input = NULL, ...)

## S4 method for signature 'sf'
gc_geom(input = NULL, group = FALSE, ...)

## S4 method for signature 'Raster'
gc_geom(input = NULL, stack = FALSE, group = FALSE, as_hex = FALSE, ...)
```

Arguments

input	the object to transform to class geom.
...	additional arguments.
group	[logical(1)] should the values of a Raster* or the attributes of MULTI* features be grouped, i.e., should the unique values be assigned into the groups table (TRUE)? The default behaviour for Raster* would be not to assign values into the group attribute table if no RAT is available and for MULTI* features it would be to keep the attributes as duplicated per-feature attributes (FALSE)?
stack	[logical(1)] should the layers of gridded objects be stacked, i.e., should several layers be stored as columns in the attribute table of features of one geom (TRUE, default), or should they be stored in (a list of) several geoms separately (FALSE)?
as_hex	[logical(1)] should the bands 'red', 'green' and 'blue' of a gridded object be transformed to hexadecimal values (TRUE), or should they be retained as columns in a stacked grid geom (FALSE, default)?.

Details

When transforming a simple feature to a geom, all MULTI* features are organised on a per feature basis, where the attribute table of features in the geom contains those variables that are valid for each feature, while the attribute table of groups contains those variables, that are unique only at the level of groups of features (i.e., at the level of MULTI* simple features). Those variables that are valid at the level of groups would be duplicated in the attribute table of features. When a MULTI* feature is transformed to a geom, the default behaviour is to copy the simple feature as closely as

possible. However, to reduce the object size (and improve its' organisation), it is possible to assign the attributes of groups into the attribute table of groups of the geom by setting `group = TRUE`.

When transforming a Raster* (or possibly other gridded classes) with several layers to a geom, the layers are by default organised into a list with a layer per list item. However, when several layers contain fundamentally the same data (i.e., values that are associated to the same groups), layers could be stacked `stack = TRUE`, because they share the same group attribute table.

Value

an object of class `geom`

See Also

Other spatial classes: `gc_raster()`, `gc_sf()`, `gc_sp()`

Examples

```
gc_geom(input = gtSF$polygon)

gc_geom(input = gtRasters$categorical)
```

---

gc_raster *Transform a spatial object to class* `Raster*`

Description

Transform a spatial object to class `Raster*`

Usage

```
## S4 method for signature 'geom'
gc_raster(input = NULL)
```

Arguments

`input`	the object to transform to class `Raster*`.

Value

an object of class `Raster*`

See Also

Other spatial classes: `gc_geom()`, `gc_sf()`, `gc_sp()`

Examples

```
gc_raster(input = gtGeoms$grid$categorical)
```

gc_sf *Transform a spatial object to class* sf

Description

Transform a spatial object to class sf

Usage

```
## S4 method for signature 'geom'
gc_sf(input = NULL)
```

Arguments

input	the object to transform to class sf.

Value

If input is a geom and has attributes other than fid and gid, a "Simple feature collection", otherwise a "Geometry set". Several features of the geom are returned as MULTI* feature, when they have gid and optionally other attributes in common, otherwise they are returned as a single simple feature.

See Also

Other spatial classes: gc_geom(), gc_raster(), gc_sp()

Examples

```
gc_sf(input = gtGeoms$point)

gc_sf(input = gtGeoms$line)

gc_sf(input = gtGeoms$polygon)
```

gc_sp *Transform a spatial object to class* Spatial

Description

Transform a spatial object to class Spatial

Usage

```
## S4 method for signature 'geom'
gc_sp(input = NULL)
```

Arguments

input	the object to transform to class Spatial.

Value

an object of class Spatial

See Also

Other spatial classes: gc_geom(), gc_raster(), gc_sf()

Examples

```
gc_sp(input = gtGeoms$point)

gc_sp(input = gtGeoms$line)

gc_sp(input = gtGeoms$polygon)
```

---

geom-class *Geometry class (S4) and methods*

Description

A geom stores a table of points, a table of feature to which the points are associated and a table of groups, to which features are associated. A geom can be spatial (if it has a coordinate reference system assigned to it), but is not by default.

Details

A geom has one of three geometry types:

- "point", when none of the points are connected to other points,
- "line", where points with the same fid are connected following the sequence of their order, without the line closing in itself and
- "polygon", where points with the same fid are connected following the sequence of their order and the line closes in on itself due to first and last point being the same. Moreover, polygon objects can contain holes.

The data model for storing points follows the spaghetti model. Points are stored as a sequence of x and y values, associated to a feature ID. The feature ID relates coordinates to features and thus common attributes. Points and Lines are implemented straightforward in this model, but polygons, which may contain holes, are a bit trickier. In geometr they are implemented as follows:

1. All points with the same fid make up one polygon, irrespective of it containing holes or not.
2. The outer path/ring of a polygon is composed of all points until a duplicated of its first point occurs. This signals that all following points are part of another path/ring, which must be inside the outer path and which consists of all points until a duplicate of it's first point occurs.

3. This repeats until all points of the feature are processed.

Moreover, a geom has a *reference window*, which is sort of a second extent that may be bigger (or smaller) than the extent and which determines the relative position of the points when plotting.

Slots

`type [character(1)]`
: the type of feature, either "point", "line", "polygon" or "grid".

`point [data.frame(1)]`
: the fid (feature ID), x and y coordinates per point and optional arbitrary point attributes.

`feature [data.frame(1)]`
: fid (feature ID), gid (group ID) and optional arbitrary feature attributes.

`group [data.frame(1)]`
: gid (group ID) and optional arbitrary group attributes.

`window [data.frame(1)]`
: the minimum and maximum value in x and y dimension of the reference window in which the geom dwells.

`scale [character(1)]`
: whether the point coordinates are stored as "absolute" values, or "relative" to window.

`crs [character(1)]`
: the coordinate reference system in proj4 notation.

`history [list(.)]`
: a list of steps taken to derive the geom in focus.

---

geometr *geometr: Generate and Modify Interoperable Geometric Shapes*

Description

The geometr package provides tools that generate and process easily accessible and tidy geometric shapes (of class geom). Moreover, it aims to improve interoperability of spatial and other geometric classes. Spatial classes are typically a collection of geometric shapes (or their vertices) that are accompanied by various metadata (such as attributes and a coordinate reference system). Most spatial classes are thus conceptually quite similar, yet a common standard lacks for accessing features, vertices or the metadata. Geometr fills this gap by providing tools

- that produce an identical output for the same metadata of different classes (via so-called getters) and
- that use an identical input to write to various classes that originally require different input (via so-called setters).

Author(s)

Maintainer, Author: Steffen Ehrmann <steffen.ehrmann@idiv.de>

Copyright holder Dan Sunday fast point-in-polygon algorithm

See Also

- Github project: https://github.com/EhrmannS/geometr
- Report bugs: https://github.com/EhrmannS/geometr/issues

`getCols` *Get the number of columns of a spatial object.*

Description

Get the number of columns of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getCols(x)

## S4 method for signature 'geom'
getCols(x)

## S4 method for signature 'Raster'
getCols(x)

## S4 method for signature 'matrix'
getCols(x)
```

Arguments

`x`	the object from which to get the number of columns.

Value

An integer of the number of columns.

See Also

Other getters: `getCRS()`, `getExtent()`, `getFeatures()`, `getGroups()`, `getHistory()`, `getLayers()`, `getNames()`, `getPoints()`, `getRes()`, `getRows()`, `getType()`, `getWindow()`

Examples

```
getCols(x = gtGeoms$grid$continuous)

getCols(x = gtRasters$categorical)

getCols(x = matrix(0, 3, 5))
```

getCRS *Get the coordinate reference system of a spatial object.*

Description

Get the coordinate reference system of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getCRS(x)

## S4 method for signature 'geom'
getCRS(x)

## S4 method for signature 'Spatial'
getCRS(x)

## S4 method for signature 'sf'
getCRS(x)

## S4 method for signature 'Raster'
getCRS(x)
```

Arguments

x	the object from which to extract the coordinate reference system.

Value

The coordinate reference system of x given as proj4string.

See Also

Other getters: getCols(), getExtent(), getFeatures(), getGroups(), getHistory(), getLayers(), getNames(), getPoints(), getRes(), getRows(), getType(), getWindow()

Examples

```
getCRS(x = gtGeoms$grid$continuous)

library(sf)
nc_sf <- st_read(system.file("shape/nc.shp", package="sf"), quiet = TRUE)
getCRS(nc_sf)

getCRS(x = gtRasters$categorical)
```

getExtent *Get the extent (bounding box) of a spatial object.*

Description

Get the extent (bounding box) of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getExtent(x)

## S4 method for signature 'geom'
getExtent(x)

## S4 method for signature 'Spatial'
getExtent(x)

## S4 method for signature 'sf'
getExtent(x)

## S4 method for signature 'Raster'
getExtent(x)

## S4 method for signature 'matrix'
getExtent(x)
```

Arguments

x	the object from which to derive the extent.

Value

A tibble of the lower left and upper right corner coordinates of the extent of x. This table two columns (x and y) and two rows (minimum and maximum).

See Also

Other getters: getCRS(), getCols(), getFeatures(), getGroups(), getHistory(), getLayers(), getNames(), getPoints(), getRes(), getRows(), getType(), getWindow()

Examples

```
getExtent(gtGeoms$polygon)

getExtent(x = gtSP$SpatialPolygons)

getExtent(x = gtSF$multilinestring)
```

```
getExtent(x = gtRasters$categorical)

getExtent(x = matrix(0, 3, 5))
```

`getFeatures` *Get the table of feature attributes*

Description

Get tabular information of the attributes of features.

Usage

```
## S4 method for signature 'ANY'
getFeatures(x)

## S4 method for signature 'geom'
getFeatures(x)

## S4 method for signature 'Spatial'
getFeatures(x)

## S4 method for signature 'sf'
getFeatures(x)

## S4 method for signature 'Raster'
getFeatures(x)

## S4 method for signature 'matrix'
getFeatures(x)
```

Arguments

x	the object from which to derive the attribute table.

Details

This table contains at least the column 'fid'. In case x has any typ other than 'grid', it contains also the column 'gid' and in case it has type 'grid', it also contains the column 'values'.

Value

A tibble (or a list of tibbles per layer) of the feature attributes of x.

See Also

Other getters: `getCRS()`, `getCols()`, `getExtent()`, `getGroups()`, `getHistory()`, `getLayers()`, `getNames()`, `getPoints()`, `getRes()`, `getRows()`, `getType()`, `getWindow()`

Examples

```
getFeatures(x = gtGeoms$polygon)

getFeatures(x = gtRasters)
```

`getGroups` *Get the table of group attributes*

Description

Get the table of group attributes

Usage

```
## S4 method for signature 'ANY'
getGroups(x)

## S4 method for signature 'geom'
getGroups(x)

## S4 method for signature 'Raster'
getGroups(x)
```

Arguments

`x`	the object from which to derive the attribute table.

Details

This table contains at least the column 'gid'.

When this function is called on "ANY" object, it is first tested whether that object has features (`getFeatures`), from which the groups can be reconstructed. If this is not the case, `NULL` is returned.

Value

A tibble (or a list of tibbles per layer) of the group attributes of `x`.

See Also

Other getters: `getCRS()`, `getCols()`, `getExtent()`, `getFeatures()`, `getHistory()`, `getLayers()`, `getNames()`, `getPoints()`, `getRes()`, `getRows()`, `getType()`, `getWindow()`

Examples

```
getGroups(x = gtGeoms$polygon)

# for raster objects, groups are pixels with the same value and their
# attributes are in the raster attribute table (RAT)
getGroups(x = gtRasters)
```

---

getHistory *Get the history of a spatial object.*

Description

Get the history of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getHistory(x)

## S4 method for signature 'geom'
getHistory(x)

## S4 method for signature 'Raster'
getHistory(x)
```

Arguments

x	the object from which to derive the history.

Value

A list of the events that lead to x.

See Also

Other getters: getCRS(), getCols(), getExtent(), getFeatures(), getGroups(), getLayers(), getNames(), getPoints(), getRes(), getRows(), getType(), getWindow()

Examples

```
library(tibble)
library(magrittr)

geom <- tibble(x = c(40, 70, 70, 50),
               y = c(40, 40, 60, 70)) %>%
  gs_polygon() %>%
  gt_reflect(angle = 45)
getHistory(x = geom)

getHistory(x = gtRasters)
```

getLayers *Get a specific layer of a spatial object.*

Description

Get a specific layer of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getLayers(x)

## S4 method for signature 'geom'
getLayers(x)

## S4 method for signature 'Spatial'
getLayers(x)

## S4 method for signature 'sf'
getLayers(x)

## S4 method for signature 'Raster'
getLayers(x)

## S4 method for signature 'matrix'
getLayers(x)
```

Arguments

x	the object from which to get the layer.

Value

A list of the layers of x. Each list-item hast the result of getNames(x) as name.

See Also

Other getters: getCRS(), getCols(), getExtent(), getFeatures(), getGroups(), getHistory(), getNames(), getPoints(), getRes(), getRows(), getType(), getWindow()

Examples

```
getLayers(x = gtRasters)
```

getNames *Get the name(s) of a spatial object.*

Description

Get the name(s) of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getNames(x)

## S4 method for signature 'geom'
getNames(x)

## S4 method for signature 'sf'
getNames(x)

## S4 method for signature 'Raster'
getNames(x)
```

Arguments

x	the object from which to get the name.

Value

A vector of the names of x.

See Also

Other getters: getCRS(), getCols(), getExtent(), getFeatures(), getGroups(), getHistory(), getLayers(), getPoints(), getRes(), getRows(), getType(), getWindow()

Examples

```
getNames(x = gtGeoms$grid$continuous)

getNames(x = gtSF$polygon)

getNames(x = gtRasters)
```

getPoints *Get the table of point coordinates*

Description

Get tabular information of the point coordinates.

Usage

```
## S4 method for signature 'ANY'
getPoints(x)

## S4 method for signature 'geom'
getPoints(x)

## S4 method for signature 'Spatial'
getPoints(x)

## S4 method for signature 'sf'
getPoints(x)

## S4 method for signature 'Raster'
getPoints(x)

## S4 method for signature 'matrix'
getPoints(x)
```

Arguments

x	the object from which to derive the point coordinates.

Details

This table contains three columns (x, y and fid) and as many rows as there are points. In case x is a polygon, the last point of each distinct feature is a duplicate of its first point. In case x has the type 'grid', all layers are summarised into one tibble, as several layers must have the same extent and resolution, so that each point occurrs in each layer, merely with a different, layer-specific value.

Value

A tibble of the point coordinates of x.

See Also

Other getters: getCRS(), getCols(), getExtent(), getFeatures(), getGroups(), getHistory(), getLayers(), getNames(), getRes(), getRows(), getType(), getWindow()

Examples

```
getPoints(x = gtGeoms$polygon)

getPoints(x = gtGeoms$point)

# for raster objects, the @point slot is extracted from its compact form
gtGeoms$grid$continuous@point

getPoints(x = gtGeoms$grid$continuous)
```

`getRes` *Get the spatial resolution of a spatial object.*

Description

Get the spatial resolution of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getRes(x)

## S4 method for signature 'geom'
getRes(x)

## S4 method for signature 'Raster'
getRes(x)

## S4 method for signature 'matrix'
getRes(x)
```

Arguments

x	the object from which to derive the resolution.

Value

A vector of two values of the spatial resolution of x in x and y dimension.

See Also

Other getters: `getCRS()`, `getCols()`, `getExtent()`, `getFeatures()`, `getGroups()`, `getHistory()`, `getLayers()`, `getNames()`, `getPoints()`, `getRows()`, `getType()`, `getWindow()`

Examples

```
getRes(x = gtGeoms$grid$continuous)

getRes(x = gtRasters$categorical)

getRes(x = matrix(0, 3, 5))
```

`getRows` *Get the number of rows of a spatial object.*

Description

Get the number of rows of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getRows(x)

## S4 method for signature 'geom'
getRows(x)

## S4 method for signature 'Raster'
getRows(x)

## S4 method for signature 'matrix'
getRows(x)
```

Arguments

`x`	the object from which to get the number of rows.

Value

An integer of the number of rows.

See Also

Other getters: `getCRS()`, `getCols()`, `getExtent()`, `getFeatures()`, `getGroups()`, `getHistory()`, `getLayers()`, `getNames()`, `getPoints()`, `getRes()`, `getType()`, `getWindow()`

Examples

```
getRows(x = gtGeoms$grid$continuous)

getRows(x = gtRasters$categorical)

getRows(x = matrix(0, 3, 5))
```

getType *Get the type of a spatial object.*

Description

Get the type of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getType(x)

## S4 method for signature 'geom'
getType(x)

## S4 method for signature 'Spatial'
getType(x)

## S4 method for signature 'sf'
getType(x)

## S4 method for signature 'Raster'
getType(x)

## S4 method for signature 'matrix'
getType(x)
```

Arguments

x	the object for which to determine the type.

Value

A vector of two values of the geometry type (point/line/polygon/grid) and the specific main type/class of x.

See Also

Other getters: getCRS(), getCols(), getExtent(), getFeatures(), getGroups(), getHistory(), getLayers(), getNames(), getPoints(), getRes(), getRows(), getWindow()

Examples

```
getType(x = gtGeoms$point)

getType(x = gtSP$SpatialPolygons)

getType(x = gtSF$multiline)
```

```
getType(x = gtRasters$categorical)

getType(x = matrix(0, 3, 5))
```

`getWindow` *Get the reference window of a spatial object.*

Description

Get the reference window of a spatial object.

Usage

```
## S4 method for signature 'ANY'
getWindow(x)

## S4 method for signature 'geom'
getWindow(x)

## S4 method for signature 'Spatial'
getWindow(x)

## S4 method for signature 'sf'
getWindow(x)

## S4 method for signature 'Raster'
getWindow(x)

## S4 method for signature 'matrix'
getWindow(x)
```

Arguments

x	the object from which to derive the reference window.

Value

A tibble of the corner coordinates of the reference window of `x`. This table two columns (x and y) and two rows (minimum and maximum).

See Also

Other getters: `getCRS()`, `getCols()`, `getExtent()`, `getFeatures()`, `getGroups()`, `getHistory()`, `getLayers()`, `getNames()`, `getPoints()`, `getRes()`, `getRows()`, `getType()`

Examples

```
getWindow(x = gtGeoms$line)

getWindow(x = gtSP$SpatialLines)

getWindow(x = gtSF$multilinestring)

getWindow(x = gtRasters$categorical)

getWindow(x = matrix(0, 3, 5))
```

---

gs_line *Create a line* geom

Description

Create a line geometry (of class geom) either by specifying anchor values or by sketching it.

Usage

```
gs_line(anchor = NULL, window = NULL, features = 1, vertices = 2, ...)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the geom from. It must include column names x, y and optionally a custom fid.
window	[data.frame(1)] in case the reference window deviates from the bounding box of anchor (minimum and maximum values), specify this here.
features	[integerish(1)] number of lines to create.
vertices	[integerish(.)] number of vertices per line; will be recycled if it does not have as many elements as specified in features.
...	[various] graphical parameters to gt_locate, in case points are sketched; see gpar

Details

The argument anchor indicates how the geom is created:

- if anchor is set, the geom is created parametrically from the points given in anchor,
- if it is not set either window or a default window between 0 and 1 is opened to sketch the geom.

Value

A geom.

See Also

Other geometry shapes: gs_point(), gs_polygon(), gs_random()

Examples

```
# 1. create a line programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))

# if no window is set, the bounding box will be set as window
(aGeom <- gs_line(anchor = coords))

# the vertices are plottet relative to the window
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aLine <- gs_line(anchor = coords, window = window)
visualise(aLine, linecol = "green")

# when a geom is used in 'anchor', its properties are passed on
aGeom <- setWindow(x = aGeom, to = window)
aLine <- gs_line(anchor = aGeom)
visualise(aLine, linecol = "deeppink")

# 2. sketch a line
if(dev.interactive()){
  aLine <- gs_line(vertices = 4)
  visualise(aLine, linecol = "orange", linewidth = 5, new = FALSE)
}
```

---

gs_point *Create a point* geom

Description

Create a point geometry (of class geom) either by specifying anchor values or by sketching it.

Usage

```
gs_point(anchor = NULL, window = NULL, vertices = 1, ...)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the geom from. It must include column names x, y and optionally a custom fid.

window	[data.frame(1)] in case the reference window deviates from the bounding box of anchor (minimum and maximum values), specify this here.
vertices	[integer(1)] number of vertices.
...	[various] graphical parameters to gt_locate, in case points are sketched; see gpar

Details

The argument anchor indicates how the geom is created:

- if anchor is set, the geom is created parametrically from the points given in anchor,
- if it is not set either window or a default window between 0 and 1 is opened to sketch the geom.

Value

A geom.

See Also

Other geometry shapes: gs_line(), gs_polygon(), gs_random()

Examples

```
# 1. create points programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))

# if no window is set, the bounding box will be set as window
(aGeom <- gs_point(anchor = coords))

# the vertices are plottet relative to the window
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
points <- gs_point(anchor = coords, window = window)
visualise(points, linecol = "green")

# when a geom is used in 'anchor', its properties are passed on
aGeom <- setWindow(x = aGeom, to = window)
points <- gs_point(anchor = aGeom)
visualise(points)

# 2. sketch two points
if(dev.interactive()){
  points <- gs_point(vertices = 2)
  visualise(points, linecol = "green", pointsymbol = 5, new = FALSE)
}
```

gs_polygon *Create a polygon* geom

Description

Create any (regular) polygon geometry (of class geom) either by specifying anchor values or by sketching it.

Usage

```
gs_polygon(
  anchor = NULL,
  window = NULL,
  features = 1,
  vertices = 3,
  regular = FALSE,
  ...
)

gs_triangle(anchor = NULL, window = NULL, features = 1, ...)

gs_square(anchor = NULL, window = NULL, features = 1, ...)

gs_rectangle(anchor = NULL, window = NULL, features = 1, ...)

gs_hexagon(anchor = NULL, window = NULL, features = 1, ...)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the geom from. It must include column names x, y and optionally a custom fid.
window	[data.frame(1)] in case the reference window deviates from the bounding box of anchor (minimum and maximum values), specify this here.
features	[integerish(1)] number of polygons to create.
vertices	[integerish(.)] number of vertices per polygon; will be recycled if it does not have as many elements as specified in features.
regular	[logical(1)] should the polygon be regular, i.e. point symmetric (TRUE) or should the vertices be selected as provided by anchor (FALSE, default)?
...	[various] graphical parameters to gt_locate, in case points are sketched; see gpar

Details

The argument anchor indicates how the geom is created:

- if anchor is set, the geom is created parametrically from the points given in anchor,
- if it is not set either window or a default window between 0 and 1 is opened to sketch the geom.

The argument regular determines how the vertices provided in anchor or via template are transformed into a polygon:

- if regular = FALSE the resulting polygon is created from all vertices in anchor,
- if regular = TRUE, only the first two vertices are considered, as center and indicating the distance to the (outer) radius.

Value

A geom.

Functions

- gs_triangle: wrapper of gs_polygon where vertices = 3 and regular = TRUE.
- gs_square: wrapper of gs_polygon where vertices = 4 and regular = TRUE.
- gs_rectangle: wrapper of gs_polygon where vertices = 2, regular = FALSE and the two complementing corners are derived from the two given opposing corners.
- gs_hexagon: wrapper of gs_polygon where vertices = 6 and regular = TRUE.

See Also

Other geometry shapes: gs_line(), gs_point(), gs_random()

Examples

```
# 1. create a polygon programmatically
coords <- data.frame(x = c(0, 40, 40, 0),
                     y = c(0, 0, 40, 40))

# if no window is set, the bounding box will be set as window
aGeom <- gs_polygon(anchor = coords)
visualise(aGeom)

# derive a regular polygon from the coordinates
aPolygon <- gs_polygon(anchor = coords, vertices = 6, regular = TRUE)
visualise(aPolygon, linecol = "green")
visualise(aGeom, new = FALSE)

# the vertices are plottet relative to the window
window <- data.frame(x = c(-50, 50),
                     y = c(-50, 50))
aPolygon <- setWindow(x = aPolygon, to = window)
visualise(aPolygon, fillcol = "deeppink")
```

```
# using a geom as anchor retains its properties (such as the window)
aRectangle <- gs_rectangle(anchor = aPolygon)
visualise(aRectangle, new = FALSE)

# 2. sketch a hexagon
if(dev.interactive()){
  aHexagon <- gs_hexagon(features = 1)
  visualise(aHexagon, linecol = "deeppink", linetype = 2, new = FALSE)
}
```

---

gs_random *Create a* geom *randomly*

Description

This function creates a random geometry

Usage

```
gs_random(type = "point", window = NULL, vertices = NULL)
```

Arguments

type	[character(1)] Either one of the three main feature types "point", "line" or "polygon", "random", or more specifically one of their subtypes, e.g. "hexagon" (subtypes currently not yet supported).
window	[data.frame(1)] in case the reference window deviates from the bounding box [0, 1] (minimum and maximum values), specify this here.
vertices	[integerish(1)] the number of vertices the geometry should have; only meaningful if type does not indicate the number of vertices already. If left at NULL the minimum number of vertices for the geom type, i.e. 1 for point, 2 for line and 3 for polygon.

Value

A geom.

See Also

Other geometry shapes: gs_line(), gs_point(), gs_polygon()

Examples

```
# create a random polygon with five vertices
set.seed(1)
someGeom <- gs_random(type = "polygon", vertices = 5)
visualise(geom = someGeom)

# in case template is given, this serves as source for the window extent
visualise(geom = gs_random(), new = FALSE, linecol = "red")
```

---

gs_tiles *Create a regular tiling* geom

Description

Create a regular tiling polygon geometry (of class geom) for the extent of an anchor value.

Usage

```
gs_tiles(anchor = NULL, width = NULL, pattern = "squared", centroids = FALSE)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the tiling geom from. It must include column names x, y and optionally a custom fid.
width	[numeric(1)] the width (which does not correspond to the height in case of pattern = "hexagonal") of a tile.
pattern	[character(1)] pattern of the tiling. Possible options are "squared" (default) or "hexagonal".
centroids	[logical(1)] should the centroids of the tiling be returned (TRUE) or should the tiling be returned (FALSE, default)?

Details

When deriving a regular tiling for a prescribed window, there is only a limited set of legal combinations of cells in x and y dimension. For instance, a window of 100 by 100 can't comprise 10 by 5 squares of side-length/width 10, because then the y-dimension wouldn't be fully covered. The same is true for hexagonal and triangular tilings.

Value

A geom.

See Also

Other tilings: gs_voronoi()

Examples

```
# create a squared tiling
aWindow <- data.frame(x = c(-180, 180),
                      y = c(-60, 80))
tiles <- gs_tiles(anchor = aWindow, width = 10)
visualise(`10° world tiles` = tiles)

# create a hexagonal tiling on top of a geom
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_polygon(anchor = coords, window = window)
visualise(`honeycomb background` = aGeom)
hex <- gs_tiles(anchor = aGeom, width = 8, pattern = "hexagonal")
visualise(hex, linecol = "deeppink", new = FALSE)
```

---

gs_voronoi *Create a voronoi tiling* geom

Description

Create a voronoi tiling geom

Usage

```
gs_voronoi(anchor = NULL, window = NULL, features = 3, ...)
```

Arguments

anchor	[geom(1)|data.frame(1)] Object to derive the geom from. It must include column names x, y and optionally a custom fid.
window	[data.frame(1)] in case the reference window deviates from the bounding box of anchor (minimum and maximum values), specify this here.
features	[integerish(1)] number of tiles to sketch.
...	[various] graphical parameters to gt_locate, in case the tiling is sketched; see gpar.

Value

A geom.

See Also

Other tilings: gs_tiles()

Examples

```
# 1. create voronoi polygons programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))
window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
aGeom <- gs_point(anchor = coords, window = window)
visualise(voronoi = aGeom, linecol = "deeppink")

tiles <- gs_voronoi(anchor = aGeom)
visualise(tiles, new = FALSE)

# 2. sketch a voronoi polygon
if(dev.interactive()){
  tiles <- gs_voronoi()
  visualise(tiles, new = FALSE)
}
```

---

gtGeoms *Example* geom *objects*

Description

A set of five geometries.

Usage

```
gtGeoms
```

Format

The list contains five objects of class geom, a `point`, `line` and `polygon` object and two `grid` geoms, one with categorical data and one with continuous data. They are mostly used in the example and test-sections of this package.

---

gtRasters *Example* RasterStack *object*

Description

A set of two conceptually different types of raster.

Usage

```
gtRasters
```

Format

The object of class RasterStack has no projection and is a RasterStack object of 56 by 60 cells. The first raster represents land-use classes and the second raster contains a continuous scale of vegetation cover.

---

gtSF *Example* sf *objects*

Description

A set of six sp objects

Usage

```
gtSF
```

Format

The list contains six objects of class sf, a POINT, a MULTIPOINT, a LINESTRING, a MULTILINESTRING, a POLYGON, and a MULTIPOLYGON object. They are mostly used in the example and test-sections of this package.

---

gtSP *Example* Spatial *objects*

Description

A set of four sp objects

Usage

```
gtSP
```

Format

The list contains four objects of class Spatial, a SpatialPoints, a SpatialMultiPoints, a, SpatialLines and a SpatialPolygons object. They are mostly used in the example and test-sections of this package.

gtTheme *Default visualising theme*

Description

Default visualising theme

Usage

```
gtTheme
```

Format

An object of class gtTheme of length 1.

gtTheme-class *Theme class (S4) and methods*

Description

An gtTheme stores a theme to visualise vector and raster objects. It is recommended to use setTheme to modify a gtTheme, because it carries out all the checks and makes sure that names of the parameters are properly matched.

Slots

`title` [named `list(3)`]
: properties of the title.

`box` [named `list(5)`]
: properties of the bounding box.

`xAxis` [named `list(5)`]
: properties of the x-axis, its labels and ticks.

`yAxis` [named `list(5)`]
: properties of the y-axis, its labels and ticks.

`grid` [named `list(5)`]
: properties of the major and minor grid.

`legend` [named `list(9)`]
: properties of the legend, its title, labels, ticks and bounding box.

`scale` [names `list(6)`]
: properties of scaling parameters to attributes/variables.

`parameters` [named `list(7)`]
: parameters of the plot object.

gt_filter *Subset a geometric object using column values*

Description

This function allows to subset any geometric object for which all required getters are available.

Usage

```
gt_filter(obj, ..., update = TRUE)
```

Arguments

obj	[geometric object(1)] the object to derive a subset from.
...	subset based on logical predicates defined in terms of the columns in x (of both, points, features and groups). Multiple conditions are combined with &. Only rows where the condition evaluates to TRUE are kept.
update	[logical(1)] whether or not to update the window slot after deriving the subset.

Value

geom of the subset of obj.

See Also

Other geometry tools: gt_locate(), gt_pull(), gt_reflect(), gt_rotate(), gt_scale(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
# get a subset of a geom
gt_filter(gtGeoms$point, y < -10)

# get a subset of an sf-object
gt_filter(obj = gtSF$multilinestring, a == 1)
```

gt_locate *Locate (and identify) clicks*

Description

Click into a plot to get the location or identify values

Usage

```
gt_locate(
  samples = 1,
  panel = NULL,
  identify = FALSE,
  snap = FALSE,
  raw = FALSE,
  show = TRUE,
  ...
)
```

Arguments

samples	[integerish(1)] the number of clicks.
panel	[character(1)] the panel in which to locate (i.e. the title shown over the plot).
identify	[logical(1)] get the raster value or geom ID at the sampled location (TRUE) or merely the location (FALSE, default).
snap	[logical(1)] should the returned value(s) be set to the nearest raster cell's center (TRUE) or should they remain the selected, "real" value (FALSE, default)?
raw	[logical(1)] should the complete statistics about the clicks be returned (TRUE), or should only the basic output be returned (FALSE, default)?
show	[logical(1)] should information be plotted (TRUE), or should they merely be returned to the console (FALSE, default)?
...	[various] graphical parameters of the objects that are created when show = TRUE.

Value

A tibble of the selected locations and, if identify = TRUE, the respective values. If show = TRUE the values are also shown in the plot.

See Also

Other geometry tools: gt_filter(), gt_pull(), gt_reflect(), gt_rotate(), gt_scale(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
if(dev.interactive()){

  # locate coordinates with geoms
  visualise(geom = gtGeoms$polygon)
  gt_locate(samples = 2)

  # locate or identify values with rasters
  visualise(raster = gtRasters$continuous)
  gt_locate(identify = TRUE, snap = TRUE)

  # with several panels, specify a target
  visualise(gtRasters)
  gt_locate(samples = 4, panel = "categorical",
            snap = TRUE, identify = TRUE)

}
```

---

gt_pull *Extract a single column from a geometric object*

Description

This function allows to extract a specific column from any geometric object for which all required getters are available and thus reflects the base function $.

Usage

```
gt_pull(obj, var)
```

Arguments

obj	[geometric object(1)] the object to pull a column from.
var	[character(1)] name of the variable to pull.

Details

This function searches for var by first looking in the groups, then the features and finally the points of obj. This results always in an output that is limited to the unique cases of var. In case you want the explicit values of, for instance, fid in obj@points, you have to extract points and then use pull on the result.

Value

vector of the column specified in `var`.

See Also

Other geometry tools: `gt_filter()`, `gt_locate()`, `gt_reflect()`, `gt_rotate()`, `gt_scale()`, `gt_skew()`, `gt_stretch()`, `gt_translate()`

Examples

```
# pull values from a geom (there are two features, thus two values) ...
gt_pull(gtGeoms$point, "fid")

# pull from a Raster* with RAT
gt_pull(gtGeoms$grid$categorical, "cover")

# pull from an sf-object
gt_pull(gtSF$point, "a")
```

---

gt_reflect *Reflect geometric objects*

Description

Reflect geometric objects across a reflection axis.

Usage

```
gt_reflect(obj, angle, fid = NULL, update = TRUE)
```

Arguments

`obj`	[`geometric object(1)`] the object to reflect.
`angle`	[`numeric(1)`] the counter-clockwise angle by which the reflection axis shall be rotated (can be negative to rotate clockwise).
`fid`	[`integerish(.)`] if only a subset of features shall be rotated, specify that here.
`update`	[`logical(1)`] whether or not to update the window slot after rotation.

Details

The reflection axis is a straight line that goes through the plot origin with the given angle, where positive angles open towards the positive y-axis and negative angles open up towards the negative y-axis.

Value

geom of the reflected obj.

See Also

Other geometry tools: gt_filter(), gt_locate(), gt_pull(), gt_rotate(), gt_scale(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
# reflect several features
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_reflect(obj = gtGeoms$polygon, angle = 45,
                      update = FALSE)
visualise(newPoly, linecol = "green", new = FALSE)

# reflect a single feature
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_reflect(obj = gtGeoms$polygon, angle = 90, fid = 2,
                      update = FALSE)
visualise(newPoly, linecol = "green", new = FALSE)
```

---

gt_rotate *Rotate geometric objects*

Description

Rotate geometric objects by a certain angle about center coordinates

Usage

```
gt_rotate(obj, x = NULL, y = NULL, angle = NULL, fid = NULL, update = TRUE)
```

Arguments

obj	[geometric object(1)] the object to rotate.
x	[numeric(1)] the x position(s) to rotate about.
y	[numeric(1)] the y position(s) to rotate about.
angle	[numeric(1)] the counter-clockwise angle by which geom shall be rotated (can be negative to rotate clockwise).
fid	[integerish(.)] if only a subset of features shall be rotated, specify that here.
update	[logical(1)] whether or not to update the window slot after rotation.

Value

geom of the rotated obj.

See Also

Other geometry tools: gt_filter(), gt_locate(), gt_pull(), gt_reflect(), gt_scale(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
# rotate all geoms
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_rotate(obj = gtGeoms$polygon, x = 0, y = 0, angle = 135,
                     update = FALSE)
visualise(geom = newPoly, linecol = "green", new = FALSE)

# rotate a single geom
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_rotate(obj = gtGeoms$polygon, x = -10, y = 0, angle = -180,
                     update = FALSE, fid = 2)
visualise(geom = newPoly, linecol = "green", new = FALSE)

# rotate different geoms about different centers by different angles
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_rotate(obj = gtGeoms$polygon,
                     x = c(0, -10),
                     y = c(-10, 0),
                     angle = c(75, -135),
                     update = FALSE)
visualise(geom = newPoly, linecol = "green", new = FALSE)
```

---

gt_scale *Scale geometric objects*

Description

Scale the vertex values of geometric objects to a values range.

Usage

```
gt_scale(obj, range = NULL, fid = NULL, update = TRUE)
```

Arguments

obj	[geometric object(1)] the object to be scaled.
range	[data.frame(2)] vector of length two for both of the x and y dimension to which the values should be scaled.

fid	[integerish(.)] if only a subset of features shall be scaled, specify that here.
update	[logical(1)] whether or not to update the window slot after scaling.

Value

geom of the scaled obj.

See Also

Other geometry tools: gt_filter(), gt_locate(), gt_pull(), gt_reflect(), gt_rotate(), gt_skew(), gt_stretch(), gt_translate()

Examples

```
# rescale to values between -10 and 10
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_scale(obj = gtGeoms$polygon, update = FALSE,
                    range = data.frame(x = c(-10, 10), y = c(-10, 10)))
visualise(geom = newPoly, linecol = "green", new = FALSE)

# rescale a single feature
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_scale(obj = gtGeoms$polygon, update = FALSE, fid = 2,
                    range = data.frame(x = c(-10, 10), y = c(-10, 10)))
visualise(geom = newPoly, linecol = "green", new = FALSE)
```

---

gt_skew *Skew geometric objects*

Description

Skew geometric objects by a shear factor in x and y dimension.

Usage

```
gt_skew(obj, x = NULL, y = NULL, fid = NULL, update = TRUE)
```

Arguments

obj	[geometric object(1)] the object to skew.
x	[numeric(1)] the shear factor in x dimension.
y	[numeric(1)] the shear factor in y dimension.

fid	[integerish(.)] if only a subset of features shall be skewed, specify that here.
update	[logical(1)] whether or not to update the window slot after skewing.

Value

geom of the skewed obj.

See Also

Other geometry tools: gt_filter(), gt_locate(), gt_pull(), gt_reflect(), gt_rotate(), gt_scale(), gt_stretch(), gt_translate()

Examples

```
# skew several features
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_skew(obj = gtGeoms$polygon, x = 0.5, update = FALSE)
visualise(geom = newPoly, linecol = "green", new = FALSE)

# skew a single feature
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_skew(obj = gtGeoms$polygon, x = 0.5, y = .7, fid = 2,
                   update = FALSE)
visualise(newPoly, linecol = "green", new = FALSE)
```

---

gt_stretch *Stretch geometric objects*

Description

Stretch geometric objects by a scale factor in x and y dimension.

Usage

```
gt_stretch(obj, x = NULL, y = NULL, fid = NULL, update = TRUE)
```

Arguments

obj	[geometric object(1)] the object to stretch.
x	[numeric(1)] the scale factor in x dimension.
y	[numeric(1)] the scale factor in y dimension.
fid	[integerish(.)] if only a subset of features shall be stretched, specify that here.

update [logical(1)]
whether or not to update the window slot after stretching.

Value

geom of the stretched obj.

See Also

Other geometry tools: gt_filter(), gt_locate(), gt_pull(), gt_reflect(), gt_rotate(), gt_scale(), gt_skew(), gt_translate()

Examples

```
# stretch several features
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_stretch(obj = gtGeoms$polygon, x = 0.5, y = 0.2,
                      update = FALSE)
visualise(geom = newPoly, linecol = "green", new = FALSE)

# stretch a single feature
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_stretch(obj = gtGeoms$polygon, x = 0.5, fid = 2, update = FALSE)
visualise(geom = newPoly, linecol = "green", new = FALSE)

# stretch feature separately
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_stretch(obj = gtGeoms$polygon,
                      x = c(0.2, 1),
                      y = c(1, 0.2),
                      update = FALSE)
visualise(geom = newPoly, linecol = "green", new = FALSE)
```

---

gt_translate *Translate geometric objects*

Description

Translate geometric objects by adding a constant in x and y dimension.

Usage

```
gt_translate(obj, x = NULL, y = NULL, fid = NULL, update = TRUE)
```

Arguments

obj	[geometric object(1)] the object to translate.
x	[numeric(1)] the translation constant (offset) in x dimension.
y	[numeric(1)] the translation constant (offset) in y dimension.
fid	[integerish(.)] if only a subset of features shall be translated, specify that here.
update	[logical(1)] whether or not to update the window slot after translation.

Value

geom of the mathematically translated obj.

See Also

Other geometry tools: gt_filter(), gt_locate(), gt_pull(), gt_reflect(), gt_rotate(), gt_scale(), gt_skew(), gt_stretch()

Examples

```
# translate several features
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_translate(obj = gtGeoms$polygon, x = 5, y = c(-10, 5),
                        update = FALSE)
visualise(geom = newPoly, linecol = "green", new = FALSE)

# translate a single feature
visualise(gtGeoms$polygon, linewidth = 3)
newPoly <- gt_translate(obj = gtGeoms$polygon, x = 5, fid = 2,
                        update = FALSE)
visualise(geom = newPoly, linecol = "green", new = FALSE)
```

---

setCRS *Set (or transform) the coordinate reference system of a spatial object.*

Description

Set (or transform) the coordinate reference system of a spatial object.

Usage

```
## S4 method for signature 'ANY'
setCRS(x)

## S4 method for signature 'geom'
setCRS(x, crs = NULL)

## S4 method for signature 'Spatial'
setCRS(x, crs = NULL)

## S4 method for signature 'sf'
setCRS(x, crs = NULL)

## S4 method for signature 'Raster'
setCRS(x, crs = NULL)
```

Arguments

x	the object for which to set the coordinate reference system.
crs	[character(1)] the coordinate reference system to set for this object.

Details

In case an object does not yet have a coordinate reference system assigned, this function simply assigns it. In case the object has already a valid crs, a transformation to the new crs will be carried out. The transformation is computed for all classes with the standard defined in the rgdal package.

Value

The object x with an assigned or transformed coordinate reference system.

See Also

Other setters: setFeatures(), setGroups(), setHistory(), setWindow()

---

setFeatures *Set a table of feature attributes*

Description

Set a table of feature attributes

Usage

```
## S4 method for signature 'ANY'
setFeatures(x)

## S4 method for signature 'geom'
setFeatures(x, table = NULL)

## S4 method for signature 'Spatial'
setFeatures(x, table = NULL)

## S4 method for signature 'sf'
setFeatures(x, table = NULL)

## S4 method for signature 'sfc'
setFeatures(x, table = NULL)
```

Arguments

x	the object to which to assign a new attribute table.
table	[data.frame(.)] the attribute table.

Value

The object x with an updated feature attribute table.

See Also

Other setters: setCRS(), setGroups(), setHistory(), setWindow()

---

setGroups *Set a table of group attributes*

Description

Set a table of group attributes

Usage

```
## S4 method for signature 'ANY'
setGroups(x)

## S4 method for signature 'geom'
setGroups(x, table = NULL)

## S4 method for signature 'RasterLayer'
setGroups(x, table = NULL)
```

Arguments

x	the object to which to assign a new attribute table.
table	[data.frame(.)] the new attribute table.

Value

The object x with an updated group attribute table.

See Also

Other setters: setCRS(), setFeatures(), setHistory(), setWindow()

---

setHistory *Set additional entries to the history of an object*

Description

Set additional entries to the history of an object

Usage

```
## S4 method for signature 'ANY'
setHistory(x)

## S4 method for signature 'geom'
setHistory(x, history = NULL)

## S4 method for signature 'RasterLayer'
setHistory(x, history = NULL)
```

Arguments

x	the object for which to set the coordinate reference system.
history	[list(1)] the history to set for this object.

Details

Both, objects of class geom and Raster* have the slot @history, which contains the provenance of that object. With setHistory, that provenance can be updated, based on the modification the object has been exposed to. This happens automatically for all geometry operations that come with geometr.

Value

The object x where the history slot has been updated.

See Also

Other setters: `setCRS()`, `setFeatures()`, `setGroups()`, `setWindow()`

`setTheme` *Create a new theme*

Description

Assign parameters in a gtTheme to create a new theme.

Usage

```
setTheme(
  from = NULL,
  title = NULL,
  box = NULL,
  xAxis = NULL,
  yAxis = NULL,
  grid = NULL,
  legend = NULL,
  scale = NULL,
  parameters = NULL
)
```

Arguments

`from`	[`gtTheme`] an gtTheme object.
`title`	[named `list(.)`] `plot` = `TRUE/FALSE`, `fontsize` and `colour` of the title.
`box`	[named `list(.)`] `plot` = `TRUE/FALSE`, `linewidth`, `linetype` and `linecol` of the bounding box (not supported recently).
`xAxis`	[named `list(.)`] `plot` = `TRUE/FALSE`, number of `bins` and `margin` of the x-axis, label [named `list(.)`] `plot` = `TRUE/FALSE`, `title`, `fontsize`, `colour` and `rotation` of the x-axis label, ticks [named `list(.)`] `plot` = `TRUE/FALSE`, `fontsize`, `colour` and number of `digits` to which to round the x-axis ticks.

yAxis [named list(.)]
plot = TRUE/FALSE, number of bins and margin of the y-axis,

label [named list(.)]
plot = TRUE/FALSE, title, fontsize, colour and rotation of the y-axis label,

ticks [named list(.)]
plot = TRUE/FALSE, fontsize, colour and number of digits to which to round the y-axis ticks.

grid [named list(.)]
plot = TRUE/FALSE, colour, linetype and linewidth of the major and minor grid and whether or not to plot the minor = TRUE/FALSE grid.

legend [named list(.)]
plot = TRUE/FALSE, number of bins, ascending = TRUE/FALSE order of values and the sizeRatio of legend/plot size,

label [named list(.)]
plot = TRUE/FALSE, fontsize and colour of the legend labels,

box [named list(.)]
plot = TRUE/FALSE, linetype, linewidth and colour of the legend box.

scale [named list(.)]
param = 'someParameter' and to = 'someAttribute' specifying which parameter shall be scale to which attribute, the value range that shall be represented by the scale, the number of bins into which the values shall be classified and the number of maxPixels.

parameters [named list(.)]
linecol, fillcol, missingcol, linetype, linewidth, pointsize and pointsymbol of the plot object.

Examples

```
input <- gtRasters$continuous
(myTheme <- setTheme(title = list(plot = FALSE)))

visualise(input, theme = myTheme)
```

---

setWindow *Set the reference window of a spatial object.*

Description

Set the reference window of a spatial object.

Usage

```
## S4 method for signature 'ANY'
setWindow(x)

## S4 method for signature 'geom'
setWindow(x, to = NULL)
```

Arguments

x	the object for which to set a new reference window.
to	any suitable data-structure that contains the minimum and maximum values in x and y-dimension to which the reference window shall be set, see Details.

Details

Possible data-structures are

- an object of class Extent,
- an object of class bbox,
- a table with two columns (named x and y) containing the minimum and maximum values for each dimension.

Value

The object x with an update reference window.

See Also

Other setters: setCRS(), setFeatures(), setGroups(), setHistory()

Examples

```
# create a polygon programmatically
coords <- data.frame(x = c(40, 70, 70, 50),
                     y = c(40, 40, 60, 70))
(aGeom <- gs_polygon(anchor = coords))
visualise(aGeom)

window <- data.frame(x = c(0, 80),
                     y = c(0, 80))
(aGeom <- setWindow(x = aGeom, to = window))

visualise(aGeom)
```

show,geom-method *Print geom in the console*

Description

Print geom in the console

Usage

```
## S4 method for signature 'geom'
show(object)
```

Arguments

object	[geom] object to show.

show,gtTheme-method *Print gtTheme in the console*

Description

Print gtTheme in the console

Usage

```
## S4 method for signature 'gtTheme'
show(object)
```

Arguments

object	[gtTheme] object to show.

visualise *Visualise geometric objects*

Description

Visualise geometric objects

Usage

```
visualise(
  ...,
  window = NULL,
  trace = FALSE,
  new = TRUE,
  clip = TRUE,
  theme = gtTheme
)
```

Arguments

...	objects to plot and optional graphical parameters.
window	[data.frame(1)] two opposing corners of a rectangle to which the plot is limited.
trace	[logical(1)] Print the provenance information of the geometric object (if available) (TRUE), or simply plot the object (FALSE, default).
new	[logical(1)] force a new plot (TRUE, default).
clip	[logical(1)] clip the plot by the plot box (TRUE, default), or plot also objects that go beyond the plot box.
theme	[list(7)] the theme from which to take graphical parameters; see setTheme for details.

Value

Returns invisibly an object of class recordedplot, see recordPlot for details (and warnings).

Examples

```
# make an empty plot
visualise()
visualise(window = getExtent(gtRasters$continuous))

coords <- data.frame(x = c(30, 60, 60, 40),
                     y = c(40, 40, 60, 70),
```

```
                     fid = 1)
(aGeom <- gs_polygon(anchor = coords))
visualise(aGeom)

win <- data.frame(x = c(0, 80),
                  y = c(0, 80))
withWindow <- setWindow(x = aGeom, to = win)
visualise(expanded = withWindow)

(aRaster <-  gtRasters$categorical)

# plot several objects together
visualise(aRaster, aGeom)

# give names
visualise(`a raster` = aRaster, `a geom` = aGeom)

# use graphical parameters ...
visualise(aGeom, linecol = "green")

# ... or a theme
visualise(aRaster, theme = setTheme(title = list(plot = FALSE)))
```

Index