Package 'spbabel'

January 8, 2019

Type Package

Version 0.5.0

Title Convert Spatial Data Using Tidy Tables

Description Tools to convert from specific formats to more general forms of spatial data. Using tables to store the actual entities present in spatial data provides flexibility, and the functions here deliberately minimize the level of interpretation applied, leaving that for specific applications. Includes support for simple features, round-trip for 'Spatial' classes and long-form tables, analogous to 'ggplot2::fortify'. There is also a more 'normal form' representation that decomposes simple features and their kin to tables of objects, parts, and unique coordinates.

URL https://mdsumner.github.io/spbabel

BugReports https://github.com/mdsumner/spbabel/issues

Depends R (>= 3.2.3)

Imports dplyr, methods, sp, tibble

Suggests testthat, ggplot2, maptools, raster, rmarkdown, knitr, covr, broom, ggpolypath, maps, sf, trip, viridis

VignetteBuilder knitr

LazyData yes

License GPL-3

RoxygenNote 6.1.1

ByteCompile TRUE

NeedsCompilation no

Author Michael D. Sumner [aut, cre]

Maintainer Michael D. Sumner <mdsumner@gmail.com>

Repository CRAN

Date/Publication 2019-01-08 18:10:10 UTC

R topics documented:

`spbabel-package` *Convert between different types of spatial objects.*

Description

Facilities for converting between different types of spatial objects, including an in-place method to modify the underlying geometry of 'Spatial' classes using data frame idioms.The spbabel package provides functions to round-trip a Spatial object to a single table and back.

Details

`sptable<-`	modify a Spatial object in-place
`sptable`	create a `tibble` from Spatial DataFrame object
`sp`	create Spatial DataFrame object from table

`as_tibble.sfg` *Individual geometries as tibbles.*

Description

Individual geometries as tibbles.

Usage

```
## S3 method for class 'sfg'
as_tibble(x)
```

Arguments

x	sf geometry of type sfg

Value

tibble

Examples

```
tibble::as_tibble(sf::st_point(c(1, 1, 1)))
```

---

feature_table *Normal form for sf*

Description

A 'feature_table' is a normal form for simple features, where all branches are recorded in one table with attributes object_, branch_, type_, parent_. All instances of parent_ are NA except for the holes in multipolygon.

Usage

```
feature_table(x, ...)
```

Arguments

x	sf object
...	ignored

Details

There is wasted information stored this way, but that's because this is intended as a lowest common denominator format.

There are three tables, objects (the feature attributes and ID), branches (the parts), coordinates (the X, Y, Z, M values).

---

holey *Multi-part, multi-holed, neighbouring, not completely topological polygons.*

Description

Created in /data-raw/ from a manual drawing built in Manifold GIS.

`map_table` *A decomposition of 'vector' map data structures to tables.*

Description

Creates a set of related tables to store the appropriate entities in spatial map data.

Usage

```
map_table(x, ...)
```

Arguments

`x`	object to tidy
`...`	arguments passed to methods

Details

The basic entities behind spatial data, and hence the "map tables" are:

vertices the positions in geometric space, e.g. x, y, z, time, long, lat, salinity etc.

branches a single connected chain of vertices, or "parts"

objects a collection of branches aligned to a row of metadata

This is the basic "topology" of traditional GIS vector data, for points, lines, polygons and their multi-counterparts. By default map_tables will produce these tables and also de-duplicated the input vertices, adding a fourth table to link vertices to branches.

Other topology types such as triangle or quad meshes can extend this four-entity model, or exist without the branches at all. See "mesh_table" ??

These are currently classed as object_table, branch_table, branch_link_vertex_table, and vertex_table. But there are no methods.

Value

list of tibbles

Examples

```
data(holey)
spholey <- sp(holey)
map_table(spholey)
```

mpoint1 *MultiPointsDataFrame data set*

Description

MultiPointsDataFrame data set

semap *"South-east" map data.*

Description

Created in /data-raw/ `semap` is the `sptable` version of some of `wrld_simpl`, and `seatt` is the matching attribute data, linked by 'object_'.

Created in /data-raw/.

Examples

```
# recreate as sp object
mp <- sp(semap, attr_tab = seatt, crs = "+proj=longlat +ellps=WGS84")
```

sf *TBD Convert from dplyr tbl form to simple features.*

Description

Not yet implemented.

Usage

```
sf(x, ...)

## S3 method for class 'data.frame'
sf(x, attr_tab = NULL, crs, ...)
```

Arguments

`x`	tibble as created by `sptable`
`...`	unused
`attr_tab`	remaining data from the attributes
`crs`	projection, defaults to `NA_character_`

Value

sf

show,SpatialPolygonsDataFrame-method
sp methods

Description

Sp methods

Usage

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

## S4 method for signature 'SpatialLinesDataFrame'
show(object)

## S4 method for signature 'SpatialPointsDataFrame'
show(object)

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

Arguments

object	Spatial object
x	Spatial object
...	ignored

sp *Convert from dplyr tbl form to Spatial*DataFrame.*

Description

Convert from dplyr tbl form to Spatial*DataFrame.

Usage

```
sp(x, ...)

## S3 method for class 'data.frame'
sp(x, attr_tab = NULL, crs, ...)
```

Arguments

x	tibble as created by sptable
...	unused
attr_tab	remaining data from the attributes
crs	projection, defaults to NA_character_

Value

Spatial*

Examples

```
library(dplyr)
semap1 <- semap  %>% dplyr::filter(y_ > -89.9999)
sp_obj <- sp(semap1, attr_tab = seatt, crs = "+proj=longlat +ellps=WGS84")
## look, seamless Antarctica!
## library(rgdal); plot(spTransform(sp_obj, "+proj=laea +lat_0=-70"))
```

---

sptable.SpatialPolygons

Convert from various forms to a table.

Description

Decompose a Spatial or sf object to a single table structured as a row for every coordinate in all the sub-geometries, including duplicated coordinates that close polygonal rings, close lines and shared vertices between objects.

Usage

```
## S3 method for class 'SpatialPolygons'
sptable(x, ...)

## S3 method for class 'SpatialLines'
sptable(x, ...)

## S3 method for class 'SpatialPointsDataFrame'
sptable(x, ...)

## S3 method for class 'SpatialMultiPointsDataFrame'
sptable(x, ...)

sptable(object) <- value

sptable(x, ...)

## S3 method for class 'trip'
map_table(x, ...)
```

Arguments

x	Spatial object
...	ignored
object	Spatial object
value	modified sptable version of object

Details

Input can be a of type sf SpatialPolygonsDataFrame, SpatialLinesDataFrame, SpatialMultiPointsDataFrame or a SpatialPointsDataFrame. For simplicity sptable and its inverses sp and sf assume that all geometry can be encoded with object, branch, island, order, x and y. and that the type of topology is identified by which of these are present.

For simple features objects with mixed types of topology the result is consistent, but probably not useful. Columns that aren't present in one type will be present, padded with NA. (This is work in progress).

This is analogous to the following but in spbabel provides a consistent way to round-trip back to Spatial classes and other forms.

- sp_tidiers (replacement of 'ggplot2::fortify').
- geom
- SpatialPolygonsDataFrame-class with its 'as(as(x, "SpatialLinesDataFrame"), "SpatialPointsDataFrame")' work flow.

Value

Spatial object

tibble with columns

- SpatialPolygonsDataFrame "object_" "branch_" "island_" "order_" "x" "y_"
- SpatialLinesDataFrame "object_" "branch_" "order_" "x_" "y_"
- SpatialPointsDataFrame "object_" x_" "y_"
- SpatialMultiPointsDataFrame "object_" "branch_" "x_" "y_"
- sf some combination of the above

Examples

```
## holey is a decomposed SpatialPolygonsDataFrame
spdata <- sp(holey)
library(sp)
plot(spdata, col = rainbow(nrow(spdata), alpha = 0.4))
points(holey$x_, holey$y_, cex = 4)
holes <- subset(holey, !island_)
## add the points that only belong to holes
points(holes$x_, holes$y_, pch = "+", cex = 2)

## manipulate based on topology
```

```
## convert to not-holes
notahole <- holes
notahole$island_ <- TRUE
#also convert to singular objects - note that this now means we have an overlapping pair of polys
#because the door had a hole filled by another object
notahole$object_ <- notahole$branch_
plot(sp(notahole), add = TRUE, col = "red")

## example using in-place modification with sptable<-
library(maptools)
data(wrld_simpl)
spdata2 <- spdata
library(dplyr)
## modify the geometry on this object without separating the vertices from the objects
sptable(spdata2) <- sptable(spdata2) %>% dplyr::mutate(x_ = x_ + 10, y_ = y_ + 5)
```

---

track *Multi-object track with x, y, z, and time.*

Description

Created in /data-raw/track.r

Index