Package ‘sfheaders’

Type Package

Title Converts Between R Objects and Simple Feature Objects

Date 2020-12-01

Version 0.4.0

Description Converts between R and Simple Feature 'sf' objects, without depending on the Simple Feature library. Conversion functions are available at both the R level, and through 'Rcpp'.

License MIT + file LICENSE

URL https://dcooley.github.io/sfheaders/

BugReports https://github.com/dcooley/sfheaders/issues

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

SystemRequirements C++11

LinkingTo geometries, Rcpp

Imports geometries (>= 0.2.0), Rcpp

Suggests covr, knitr, testthat

NeedsCompilation yes

Author David Cooley [aut, cre],
Michael Sumner [ctb]

Maintainer David Cooley <david.cooley.au@gmail.com>

Repository CRAN

Date/Publication 2020-12-01 00:50:03 UTC

R topics documented:

sfc_cast ................................................................. 2
sfc_linestring ......................................................... 3
sfc_multilinestring .................................................... 5
sfc_cast

Description

convert the input sfc to a different geometry

Usage

sfc_cast(sfc, to, close = TRUE)

Arguments

sfc geometry object to convert to a different geometry
to the geometry to convert to.
close logical indicating if polygons should be closed
Examples

df <- data.frame(
id1 = c(1,1,1,1,1,1,1,1,1,2,2,2),
id2 = c(1,1,1,1,2,2,2,2,1,1,1,1),
x = c(0,0,1,1,1,1,2,2,3,4,4,3),
y = c(0,1,1,0,1,2,2,1,3,3,4,4)
)

pt <- sfc_point(obj = df, x = "x", y = "y", z = "id1")
mpt <- sfc_multipoint(obj = df, x = "x", y = "y", multipoint_id = "id1")
ls <- sfc_linestring(obj = df, x = "x", y = "y", linestring_id = "id1")
mls <- sfc_multilinestring(obj = df, x = "x", y = "y", multilinestring_id = "id1")
p <- sfc_polygon(
  obj = df,
  x = "x",
  y = "y",
  polygon_id = "id1",
  linestring_id = "id2",
  close = FALSE
)
mp <- sfc_multipolygon(
  obj = df,
  x = "x",
  y = "y",
  multipolygon_id = "id1",
  linestring_id = "id2",
  close = FALSE
)
sfc_cast( pt, "LINESTRING" )
sfc_cast( mpt, "POLYGON" )
sfc_cast( ls, "POINT" )
sfc_cast( mls, "MULTIPOLYGON" )
sfc_cast( p, "POINT" )
sfc_cast( mp, "LINESTRING" )

Description

constructs sfc of LINESTRING objects

Usage

sfc_linestring(
  obj = NULL,
  x = NULL,
y = NULL,
z = NULL,
m = NULL,
linestring_id = NULL
)

Arguments

obj sorted matrix or data.frame
x x geometry column
y y geometry column
z z geometry column
m m geometry column
linestring_id column of ids for linestrings

Value

sfc object of LINESTRING geometries

notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.
The data.frame and matrices you send into the sfheader functions must be ordered.

Examples

x <- matrix( c(1:4), ncol = 2 )
sfc_linestring( x )

x <- data.frame( id = 1:2, x = 1:2, y = 2:1 )
sfc_linestring( x )
sfc_linestring( x, x = "x", y = "y" )
sfc_linestring( x, x = "y", y = "x" )
sfc_linestring( x, linestring_id = "id", x = "x", y = "y" )

df <- data.frame(
  id = c(1,1,1,1,2,2,2)
  , x = 1:7
  , y = 7:1
  , z = 14:8
  , m = 8:14
)

sfc_linestring(df, x = "x", y = "y", linestring_id = "id")
sfc_linestring(df, x = "x", y = "y", z = "z", linestring_id = "id")
sfc_linestring(df, x = "x", y = "y", m = "m", linestring_id = "id")
sfc_linestring(df, x = "x", y = "y", z = "z", m = "m", linestring_id = "id")
Description

constructs an sfc of MULTILINESTRING objects

Usage

sfc_multilinestring(
  obj = NULL,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  multilinestring_id = NULL,
  linestring_id = NULL
)

Arguments

obj            sorted matrix or data.frame
x              x geometry column
y              y geometry column
z              z geometry column
m              m geometry column
multilinestring_id
               column of ids for multilinestrings
linestring_id  column of ids for linestrings (within multilinestrings)

Value

sfc object of MULTILINESTRING geometries

notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.
Examples

\[
m <- \text{matrix}(c(0,0,0,0,1,1), ncol = 3 )
\]
\[
\text{sfc_multilinestring( m )}
\]

\[
m <- \text{matrix}(c(0,0,0,0,1,1,1,1,2,2,1,2), ncol = 3, byrow = \text{TRUE})
\]
\[
\text{sfc_multilinestring( obj = m )}
\]
\[
\text{sfc_multilinestring( obj = m, multilinestring_id = 1 )}
\]
\[
\text{sfc_multilinestring( obj = m, linestring_id = 1 )}
\]
\[
\text{sfc_multilinestring( obj = m, linestring_id = 1, multilinestring_id = 1 )}
\]
\[
\text{sfc_multilinestring( obj = m, x = 2, y = 3 )}
\]
\[
\text{sfc_multilinestring( obj = m, x = 1, y = 2, z = 3 )}
\]
\[
\text{sfc_multilinestring( obj = m, x = 2, y = 3, linestring_id = 1, multilinestring_id = 1 )}
\]
\[
df <- \text{data.frame(}
\quad \text{ml_id = c(1,1,1,1,1,1,1,1,2,2,2,2,2)}
\quad \text{, l_id = c(1,1,1,2,2,3,3,3,1,1,1,2,2)}
\quad \text{, x = rnorm(13)}
\quad \text{, y = rnorm(13)}
\quad \text{, z = rnorm(13)}
\quad \text{, m = rnorm(13)}
\quad \text{)}
\]
\[
\text{sfc_multilinestring( obj = df, x = "x", y = "y")}
\]
\[
\text{sfc_multilinestring( obj = df, x = "x", y = "y", z = "z")}
\]
\[
\text{sfc_multilinestring( obj = df, x = "x", y = "y", z = "z", m = "m")}
\]
\[
\text{sfc_multilinestring( obj = df, x = 2, y = 3 )}
\]
\[
\text{sfc_multilinestring( obj = df, x = 2, y = 3, z = 4 )}
\]
\[
\text{sfc_multilinestring( obj = df, x = 2, y = 3, z = 4, m = 5 )}
\]
\[
\text{sfc_multilinestring( obj = df, multilinestring_id = "ml_id", linestring_id = "l_id" )}
\]
\[
\text{sfc_multilinestring( obj = df, multilinestring_id = 1, linestring_id = 2 )}
\]

\[\text{sfc_multipoint} \quad \text{sfc MULTIPOINT}\]

Description

constructs sfc of MULTIPOINT objects

Usage

\[\text{sfc_multipoint(}\]
sfc_multipoint

    obj,
    x = NULL,
    y = NULL,
    z = NULL,
    m = NULL,
    multipoint_id = NULL
)

Arguments

obj sorted matrix or data.frame
x x geometry column
y y geometry column
z z geometry column
m m geometry column
multipoint_id column of ids for multipoints

Value

sfc object of MULTIPOINT geometries

notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Examples

    x <- matrix( c(1:4), ncol = 2 )
sfc_multipoint( x )

    x <- data.frame( id = 1:2, x = 1:2, y = 2:1 )
sfc_multipoint( x )
sfc_multipoint( x, x = "x", y = "y" )
sfc_multipoint( x, x = "y", y = "x" )
sfc_multipoint( x, multipoint_id = "id", x = "x", y = "y" )
sfc_multipolygon  sfc MULTIPOLYGON

Description

constructs an sfc of MULTIPOLYGON objects

Usage

sfc_multipolygon(
  obj = NULL,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  multipolygon_id = NULL,
  polygon_id = NULL,
  linestring_id = NULL,
  close = TRUE
)

Arguments

obj  sorted matrix or data.frame
x    x geometry column
y    y geometry column
z    z geometry column
m    m geometry column
multipolygon_id  column of ids for multipolygons
polygon_id  column of ids for polygons
linestring_id  column of ids for lines (within polygons)
close  logical indicating whether polygons should be closed. If TRUE, all polygons will be checked and force closed if possible

Value

sfc object of MULTIPOLYGON geometries

notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.
Examples

```r
m <- matrix(c(0,0,0,1,0,1,0,0,1,0,0,0), ncol = 3, byrow = TRUE )
sfc_multipolygon( m )

df <- data.frame(
  id = c(1,1,1,1,1),
  x = c(0,0,1,1,0),
  y = c(0,1,1,0,0)
)
sfc_multipolygon( df, x = "x", y = "y" )

df <- data.frame(
  id = c(1,1,1,1,2,2,2,2,2),
  x = c(0,0,1,1,0,1,2,2,1),
  y = c(0,1,0,0,1,2,1,1,1)
)
sfc_multipolygon( df, multipolygon_id = "id", polygon_id = "id", linestring_id = "id")

df <- data.frame(
  id1 = c(1,1,1,1,1,1,1,1,1,1,1,1),
  id2 = c(1,1,1,1,2,2,2,2,2),
  x = c(0,0,1,1,0,1,2,2,1,1),
  y = c(0,1,0,0,1,2,2,1,1)
)
sfc_multipolygon( df, multipolygon_id = "id1", polygon_id = "id2")

df <- data.frame(
  id1 = c(1,1,1,1,1,1,1,1,1,1,2,2,2,2),
  id2 = c(1,1,1,1,2,2,2,2,1,1,1,1),
  x = c(0,0,1,1,0,1,2,2,1,3,3,4,3),
  y = c(0,1,0,0,1,2,2,1,1,3,4,3,3)
)
sfc_multipolygon( df, multipolygon_id = "id1", polygon_id = "id2")

df <- data.frame(
  id1 = c(1,1,1,1,1,2,2,2,2),
  id2 = c(1,1,1,1,1,1,1,1,1),
  x = c(0,0,1,1,0,1,2,2,1,1),
  y = c(0,1,0,0,1,2,2,1,1)
)
sfc_multipolygon( df, multipolygon_id = "id1", polygon_id = "id2")
sfc_multipolygon( df, polygon_id = "id1", linestring_id = "id2" )
sfc_multipolygon( df, x = "x", y = "y", polygon_id = "id1" )
sfc_multipolygon( df, x = "x", y = "y", polygon_id = "id1", linestring_id = "id2" )
sfc_multipolygon( df, x = "x", y = "y", linestring_id = "id1" )
sfc_multipolygon( df, x = "x", y = "y", linestring_id = "id2" )
```
df <- data.frame(
  id1 = c('a', 'a', 'a', 'a', 'a', 'b', 'b', 'b'),
  id2 = c(1, 1, 1, 1, 1, 1, 1, 1),
  x = c(0, 0, 1, 1, 0, 1, 2, 2, 1),
  y = c(0, 1, 0, 1, 2, 2, 1, 1)
)

sfc_multipolygon(df, x = "x", y = "y", polygon_id = "id1")

---

**sfc_point**  
*sfc POINT*

**Description**
constructs *sfc* of POINT objects

**Usage**
sfc_point(obj, x = NULL, y = NULL, z = NULL, m = NULL)

**Arguments**
- **obj**  
  sorted vector, matrix or data.frame
- **x**  
  x geometry column
- **y**  
  y geometry column
- **z**  
  z geometry column
- **m**  
  m geometry column

**Value**
sfc object of POINT geometries

**notes**
sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.
The data.frame and matrices you send into the sfheader functions must be ordered.
Examples

```r
x <- c(1:3)
sfc_point(x)

x <- matrix(c(1:10), ncol = 2)
sfc_point(x)

x <- setNames(as.data.frame(x), c("x","y"))
sfc_point(x)
sfc_point(obj = x, x = "x", y = "y")
sfc_point(obj = x, x = "y", y = "x")
```

Description

constructs an sfc of POLYGON objects

Usage

```r
sfc_polygon(
  obj = NULL,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  polygon_id = NULL,
  linestring_id = NULL,
  close = TRUE
)
```

Arguments

- **obj** : sorted matrix or data.frame
- **x** : x geometry column
- **y** : y geometry column
- **z** : z geometry column
- **m** : m geometry column
- **polygon_id** : column of ids for polygons
- **linestring_id** : column of ids for lines (within polygons)
- **close** : logical indicating whether polygons should be closed. If TRUE, all polygons will be checked and force closed if possible
Value

sfc object of POLYGON geometries

notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Examples

```
m <- matrix(c(0,0,0,0,1,1), ncol = 2 )
sfc_polygon( m )

m <- matrix(c(0,0,0,0,1,1,0,0,1,1,2,2,1,2,3,1,3,2), ncol = 3, byrow = TRUE)
sfc_polygon( obj = m )
sfc_polygon( obj = m, polygon_id = 1 )
sfc_polygon( obj = m, linestring_id = 1 )

sfc_polygon( obj = m, linestring_id = 1, polygon_id = 1 )

sfc_polygon( obj = m, x = 2, y = 3 )
sfc_polygon( obj = m, x = 1, y = 2, z = 3 )
sfc_polygon( obj = m, x = 2, y = 3, linestring_id = 1, polygon_id = 1 )

df <- data.frame( ml_id = c(1,1,1,1,1,1,1,1,1,2,2,2,2,2,2),
                   l_id = c(1,1,1,2,2,2,3,3,3,1,1,1,2,2,2),
                   x = rnorm(15),
                   y = rnorm(15),
                   z = rnorm(15),
                   m = rnorm(15) )
sfc_polygon( obj = df, x = "x", y = "y")
sfc_polygon( obj = df, x = "x", y = "y", z = "z")
sfc_polygon( obj = df, x = "x", y = "y", z = "z", m = "m")

sfc_polygon( obj = df, x = 2, y = 3)
sfc_polygon( obj = df, x = 2, y = 3, z = 4)
sfc_polygon( obj = df, x = 2, y = 3, z = 4, m = 5)

sfc_polygon( obj = df, polygon_id = "ml_id", linestring_id = "l_id")
sfc_polygon( obj = df, polygon_id = 1, linestring_id = 2 )
```
sfc_to_df

Description

Converts an sfc object to a data.frame

Usage

sfc_to_df(sfc)

Arguments

sfc sfc object

Examples

x <- matrix( c(1:16), ncol = 2 )
sfc <- sfc_linestring( x )
df <- sfc_to_df( sfc )

df <- data.frame(
  ml_id = c(1,1,1,1,1,1,1,1,2,2,2,2,2),
  l_id = c(1,1,1,2,2,3,3,3,1,1,1,2,2),
  x = rnorm(13),
  y = rnorm(13),
  z = rnorm(13),
  m = rnorm(13)
)
sfc <- sfc_multilinestring( obj = df, multilinestring_id = "ml_id", linestring_id = "l_id" )
df <- sfc_to_df( sfc )

sfg_linestring

Description

constructs sfg LINESTRING object

Usage

sfg_linestring(obj, x = NULL, y = NULL, z = NULL, m = NULL)
Arguments

- **obj**: matrix or data.frame
- **x**: x geometry column
- **y**: y geometry column
- **z**: z geometry column
- **m**: m geometry column

Value

sfg object of LINESTRING geometry

Examples

```r
sfg_linestring(1:2)
sfg_linestring(1:3)
sfg_linestring(1:4)
sfg_linestring(matrix(1:24, ncol = 2))
sfg_linestring(matrix(1:24, ncol = 3))
sfg_linestring(matrix(1:24, ncol = 4))
sfg_linestring(matrix(1:24, ncol = 4), x = 3, y = 2, z = 3)
sfg_linestring(data.frame(x = 1:10, y = 11:20))
sfg_linestring(data.frame(x = 1:10, y = 11:20, z = 21:30))
sfg_linestring(data.frame(x = 1:10, y = 11:20, z = 21:30), x = "x", y = "z")
```

Description

constructs sfg MULTILINESTRING object

Usage

```r
sfg_multilinestring(
  obj,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  linestring_id = NULL
)
```
Arguments

obj  matrix or data.frame
x   x geometry column
y   y geometry column
z   z geometry column
m   m geometry column
linestring_id  column of ids for lines

Value

sfg object of MULTILINESTRING geometry

Examples

sfg_multilinestring( matrix( 1:24, ncol = 2 ) )
sfg_multilinestring( matrix( 1:24, ncol = 3 ) )
sfg_multilinestring( matrix( 1:24, ncol = 4 ) )

## different lines
m <- cbind( matrix( 1:24, ncol = 2 ), c(rep(1, 6), rep(2, 6)) )
sfg_multilinestring( obj = m, x = 1, y = 2, linestring_id = 3 )

## just specifying linestring_id will use all others as the geometries
sfg_multilinestring( obj = m, linestring_id = 3 )

df <- data.frame( x = 1:12, y = 1:12, z = 13:24, id = c(rep(1,6), rep(2,6)))
sfg_multilinestring( df, x = "x", y = "y" )
sfg_multilinestring( df, x = "x", y = "y", linestring_id = "id" )
sfg_multilinestring( df, linestring_id = "id" )

Description

constructs sfg MULTIPOINT object

Usage

sfg_multipoint(obj, x = NULL, y = NULL, z = NULL, m = NULL)
Arguments

- **obj**: matrix or data.frame
- **x**: x geometry column
- **y**: y geometry column
- **z**: z geometry column
- **m**: m geometry column

Value

sfg object of MULTIPOINT geometry

Examples

```r
sfg_multipoint(1:2)
sfg_multipoint(1:3)
sfg_multipoint(1:4)

sfg_multipoint(matrix(1:3, ncol = 3))
sfg_multipoint(data.frame(x = 1, y = 2, z = 3))

sfg_multipoint(matrix(1:4, ncol = 2))
sfg_multipoint(matrix(1:24, ncol = 2, byrow = TRUE))
sfg_multipoint(matrix(1:24, ncol = 3, byrow = TRUE))
sfg_multipoint(matrix(1:24, ncol = 4, byrow = TRUE))

sfg_multipoint(data.frame(x = 1:5, y = 1:5))

# using columns
sfg_multipoint(matrix(1:24, ncol = 4, byrow = TRUE), x = 1, y = 2)
sfg_multipoint(matrix(1:24, ncol = 4, byrow = TRUE), x = 1, y = 2, z = 3)
sfg_multipoint(matrix(1:24, ncol = 4, byrow = TRUE), x = 3, y = 4)

df <- data.frame(x = 1:5, y = 1:5, z = 11:15, m = 11:15)
sfg_multipoint(df, x = "x", y = "y")
sfg_multipoint(df, x = "x", y = "y", z = "z")
sfg_multipoint(df, x = "x", y = "y", z = "z", m = "m")
```

Description

constructs sfg MULTIPOLYGON object
**Usage**

```r
sfg_multipolygon(
  obj,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  polygon_id = NULL,
  linestring_id = NULL,
  close = TRUE
)
```

**Arguments**

- `obj` - matrix or data.frame
- `x` - x geometry column
- `y` - y geometry column
- `z` - z geometry column
- `m` - m geometry column
- `polygon_id` - column of ids for polygons (within the multipolygon)
- `linestring_id` - column of ids for lines (within polygons)
- `close` - logical indicating whether polygons should be closed. If TRUE, all polygons will be checked and force closed if possible

**Value**

`sfg` object of MULTIPOLYGON geometry

**Examples**

```r
df <- data.frame(
  polygon_id = c(rep(1, 5), rep(2, 10)),
  line_id = c(rep(1, 10), rep(2, 5)),
  x = c(0,0,1,1,0,2,2,5,5,2,3,3,4,4,3),
  y = c(0,1,1,0,0,2,5,5,2,2,3,4,4,3),
  z = c(1),
  m = c(1)
)

m <- as.matrix( df )

sfg_multipolygon( df[, c("x","y") ] )

sfg_multipolygon(
  df, x = "x", y = "y", polygon_id = "polygon_id", linestring_id = "line_id"
)

sfg_multipolygon(
```

---

This document contains the code and explanation for the function `sfg_multipolygon` in the `sfg` package. The function takes various arguments to create a multipolygon object from a given set of coordinates. The code examples demonstrate how to use the function with different sets of arguments, including the columns for x, y, z, and m coordinates, as well as the polygon and linestring ids.
sfg_point

    df, x = "x", y = "y", z = "z", polygon_id = "polygon_id", linestring_id = "line_id"
    sfg_multipolygon
    df, x = "x", y = "y", z = "z", m = "m", polygon_id = "polygon_id", linestring_id = "line_id"
    
    sfg_multipolygon( m[, c("x","y") ] )
    sfg_multipolygon( m, x = "x", y = "y", polygon_id = "polygon_id", linestring_id = "line_id"
    )
    sfg_multipolygon( m, x = "x", y = "y", z = "z", polygon_id = "polygon_id", linestring_id = "line_id"
    )
    sfg_multipolygon( m, x = "x", y = "y", z = "z", m = "m", polygon_id = "polygon_id", linestring_id = "line_id"
    )

sfg_point
sfg point

Description
constructs sfg POINT object

Usage
sfg_point(obj, x = NULL, y = NULL, z = NULL, m = NULL)

Arguments

obj matrix or data.frame
x x geometry column
y y geometry column
z z geometry column
m m geometry column

Value
sfg object of POINT geometry
**sfg_polygon**

**Examples**

```r
sfg_point( 1:2 )
sfg_point( 1:3 )
sfg_point( 1:4 )

sfg_point( matrix( 1:3, ncol = 3 ) )
sfg_point( data.frame( x = 1, y = 2, z = 3 ) )

sfg_point( data.frame( x = 1, y = 2, z = 3 ), x = "x", y = "y" )
sfg_point( data.frame( x = 1, y = 2, z = 3 ), x = 1, y = 3 )
```

**Description**

constructs sfg POLYGON object

**Usage**

```r
sfg_polygon(
  obj,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  linestring_id = NULL,
  close = TRUE
)
```

**Arguments**

- **obj**: matrix or data.frame
- **x**: x geometry column
- **y**: y geometry column
- **z**: z geometry column
- **m**: m geometry column
- **linestring_id**: column of ids for lines (within polygons)
- **close**: logical indicating whether polygons should be closed. If TRUE, all polygons will be checked and force closed if possible

**Value**

sfg object of POLYGON geometry
Examples

sfg_polygon( matrix( 1:24, ncol = 2 ) )
sfg_polygon( matrix( 1:24, ncol = 3 ) )
sfg_polygon( matrix( 1:24, ncol = 4 ) )

## different lines
m <- cbind( matrix( 1:24, ncol = 2 ), c(rep(1, 6), rep(2, 6) ) )
sfg_polygon( obj = m, x = 1, y = 2, linestring_id = 3 )

## just specifying linestring_id will use all others as the geometries
sfg_polygon( obj = m, linestring_id = 3 )

df <- data.frame( x = 1:12, y = 1:12, z = 13:24, id = c(rep(1,6), rep(2,6)))
sfg_polygon( df, x = "x", y = "y" )
sfg_polygon( df, x = "x", y = "y", linestring_id = "id" )
sfg_polygon( df, linestring_id = "id" )

Description

Converts an sfg object to a data.frame

Usage

sfg_to_df(sfg)

Arguments

sfg sfg object

Examples

sfg <- sfg_point( obj = c(1,2) )
df <- sfg_to_df( sfg )

m <- cbind( matrix( 1:24, ncol = 2 ), c(rep(1, 6), rep(2, 6) ) )
sfg <- sfg_polygon( obj = m, x = 1, y = 2, linestring_id = 3 )
df <- sfg_to_df( sfg )
sf_bbox

Description

Calculates the bounding box of coordinates. This does not read the "bbox" attribute, it re-calculates the bounding box from the geometry coordinates.

Usage

sf_bbox(obj, x = NULL, y = NULL)

Arguments

- obj: matrix, data.frame, sfg, sfc or sf object.
- x: x geometry column.
- y: y geometry column.

Examples

```r
## data.frame
df <- data.frame(
id1 = c(1,1,1,1,1,1,1,1,2,2,2,2),
id2 = c(1,1,1,1,2,2,2,2,1,1,1,1),
x = c(0,0,1,1,2,2,3,3,4,4,3,3),
y = c(0,1,1,0,1,2,2,1,3,4,4,4)
)

sf_bbox(obj = df[, c("x","y")])
sf_bbox(obj = df, x = "x", y = "y")

## sfg objects
pt <- sfg_point(obj = df[1, ], x = "x", y = "y", z = "id1")
mpt <- sfg_multipoint(obj = df, x = "x", y = "y")
ls <- sfg_linestring(obj = df, x = "x", y = "y")
mls <- sfg_multilinestring(obj = df, x = "x", y = "y")
p <- sfg_polygon(obj = df, x = "x", y = "y")
mp <- sfg_multipolygon(obj = df, x = "x", y = "y", close = FALSE)

sf_bbox(pt)
sf_bbox(mpt)
sf_bbox(ls)
sf_bbox(mls)
sf_bbox(p)
sf_bbox(mp)

## sfc objects
pt <- sfc_point(obj = df, x = "x", y = "y", z = "id1")
```
mpt <- sfc_multipoint(obj = df, x = "x", y = "y", multipoint_id = "id1")
ls <- sfc_linestring(obj = df, x = "x", y = "y", linestring_id = "id1")
mls <- sfc_multilinestring(obj = df, x = "x", y = "y", multilinestring_id = "id1")
p <- sfc_polygon(
  obj = df,
  x = "x",
  y = "y",
  polygon_id = "id1",
  linestring_id = "id2",
  close = FALSE
)
mp <- sfc_multipolygon(
  obj = df,
  x = "x",
  y = "y",
  multipolygon_id = "id1",
  linestring_id = "id2",
  close = FALSE
)
sf_bbox( pt )
sf_bbox( mpt )
sf_bbox( ls )
sf_bbox( mls )
sf_bbox( p )
sf_bbox( mp )

## sf objects
pt <- sf_point(obj = df, x = "x", y = "y", z = "id1")
mpt <- sf_multipoint(obj = df, x = "x", y = "y", multipoint_id = "id1")
ls <- sf_linestring(obj = df, x = "x", y = "y", linestring_id = "id1")
mls <- sf_multilinestring(obj = df, x = "x", y = "y", multilinestring_id = "id1")
p <- sf_polygon(
  obj = df,
  x = "x",
  y = "y",
  polygon_id = "id1",
  linestring_id = "id2",
  close = FALSE
)
mp <- sf_multipolygon(
  obj = df,
  x = "x",
  y = "y",
  multipolygon_id = "id1",
  linestring_id = "id2",
  close = FALSE
)
sf_bbox( pt )
sf_bbox( mpt )
sf_bbox( ls )
sf_bbox( mls )
sf_boxes

sf_bbox( p )
sf_bbox( mp )

## you can use it to update a bounding-box if it gets corrupted
attr( mpt, "bbox" ) <- c(1:5)
mpt ## incorrect values
attr( mpt, "bbox" ) <- sf_bbox( mpt )
mpt ## back to correct values

---

Description

returns the bounding box of each geometry

Usage

sf_boxes(obj)

Arguments

obj sf, sfc or sfg object

Examples

df <- data.frame(
id1 = c(1,1,1,1,1,1,1,1,1,2,2,2),
id2 = c(1,1,1,2,2,2,2,1,1,1),
x = c(0,0,1,1,1,1,2,2,3,4),
y = c(0,1,0,1,2,2,1,3,4,4)
)
sf_line <- sfheaders::sf_linestring(
    obj = df,
    x = "x",
    y = "y",
    linestring_id = "id1"
)
sf_boxes( sf_line )
Description

calculate the input sf to a different geometry

Usage

sf_cast(sf, to, close = TRUE)

Arguments

sf  object to convert

to  the geometry to convert to.

close  logical indicating if polygons should be closed

Examples

df <- data.frame(
  id1 = c(1,1,1,1,1,1,2,2,2,2,2,2),
  id2 = c(1,1,1,1,2,2,2,2,1,1,1,1),
  x = c(0,0,1,1,1,1,2,2,3,4,4,3),
  y = c(0,1,0,1,2,2,1,3,3,4,4,4)
)

pt <- sf_point(obj = df, x = "x", y = "y", z = "id1")
mp <- sf_multipoint(obj = df, x = "x", y = "y", multipoint_id = "id1")
ls <- sf_linestring(obj = df, x = "x", y = "y", linestring_id = "id1")
mls <- sf_multilinestring(obj = df, x = "x", y = "y", multilinestring_id = "id1")
p <- sf_polygon(
  obj = df,
  x = "x",
  y = "y",
  polygon_id = "id1",
  linestring_id = "id2",
  close = FALSE
)
mp <- sf_multipolygon(
  obj = df,
  x = "x",
  y = "y",
  multipolygon_id = "id1",
  linestring_id = "id2",
  close = FALSE
)

sf_cast( pt, "LINESTRING" )

sf_line

Helper for sf LINESTRING

Description

Constructs sf of LINESTRING objects, a helper for sf_linestring() with a simpler syntax.

Usage

sf_line(obj, keep = FALSE, list_columns = NULL)

Arguments

obj
sorted matrix or data.frame

keep
logical indicating if the non-geometry and non-id columns should be kept. if TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.

list_columns
vector of column names to turn into a list.

Value

sf object of LINESTRING geometries

Helpers

These are simpler versions of the main functions sf_point(), sf_multipoint(), sf_linestring(), sf_multilinestring(), sf_polygon(), and sf_multipolygon() for input data frame or matrix that contains columns appropriately of 'x', 'y', 'z', 'm', 'multipolygon_id', polygon_id', 'multipoint_id', 'multipolygon_id', 'multipolygon_id', 'linestring_id', 'linestring_id', 'linestring_id', 'multipoint_id'.

This puts the onus of the naming and identification of entities onto the input data set, rather than when calling the creator function. This has pros and cons, so is not necessarily always 'simpler'. Please choose the appropriate constructor for the context you have. For examples a data frame from the real world with columns 'lon', 'lat', 'line' will be best used with

sf_linestring(df,x = "lon",y = "lat",linestring_id = "line")

whereas a heavy user of sfheaders might always create a data frame with 'x', 'y', 'linestring_id' precisely because they are expecting to call sf_line(df) and no further work is required. These are very different contexts and both equally valid.
Some columns are mandatory, such as 'x' and 'y' (always), while others depend on the output type where each column for that type is mandatory. The 'z' and/or 'm' values are included for 'XYZ', 'XYM', or 'XYZM' geometry types if and as they are present.

In summary these helpers:

- do not require arguments declaring column names.
- use assumed default column names, with no variation or absence allowed for a given type.
- use z, and/or m if present.
- use close = FALSE and keep = FALSE same as proper constructors.
- unlike sf_point() sf_pt() does not accept a flat vector for a single point.
- require a matrix or data frame with complete column names.

None of the helpers allow partial name matching for column names.

**Notes**

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

**Keeping Properties**

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

**Examples**

```r
x <- cbind(x = 1:2, y = 3:4, linestring_id = 1)
sf_line( x )

x <- data.frame( linestring_id = rep(1:2, each = 2), x = 1:4, y = 4:1 )
(sfx <- sf_line( x ))

## we trivially round-trip with sf_line()
sf_line(sf_to_df(sfx))
```
sf_linestring

Description

corrects sf of LINESTRING objects

Usage

sf_linestring(
    obj = NULL,
    x = NULL,
    y = NULL,
    z = NULL,
    m = NULL,
    linestring_id = NULL,
    keep = FALSE,
    list_columns = NULL
)

Arguments

- obj: sorted matrix or data.frame
- x: x geometry column
- y: y geometry column
- z: z geometry column
- m: m geometry column
- linestring_id: column of ids for linestrings
- keep: logical indicating if the non-geometry and non-id columns should be kept. If TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.
- list_columns: vector of column names to turn into a list.

Value

- sf object of LINESTRING geometries

notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.
Keeping Properties

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

```r
x <- matrix( c(1:8), ncol = 2 )
sf_linestring( x )

x <- cbind( x, c(1,1,2,2) )
sf_linestring( obj = x, x = 1, y = 2 )
sf_linestring( obj = x, x = 1, y = 2, linestring_id = 3 )

x <- data.frame( line_id = 1:2, x = 1:2, y = 2:1 )
sf_linestring( x )
sf_linestring( x, x = "x", y = "y" )
sf_linestring( x, x = "y", y = "x" )
sf_linestring( x, linestring_id = "line_id", x = "x", y = "y" )

## keeping properties
x <- data.frame(
  line_id = c(1,1,2,2),
  x = 1:4,
  y = 4:1,
  val = letters[1:4],
  stringsAsFactors = FALSE
)

## first-row of 'val' is kept
sf_linestring( x, x = "x", y = "y", keep = TRUE )
sf_linestring( x, linestring_id = "line_id", x = "x", y = "y", keep = TRUE )

## 'val' column converted to a list
sf_linestring( x, linestring_id = "id", x = "x", y = "y", keep = TRUE, list_columns = "val" )
```

### sf_mline

**Helper for sf MULTILINESTRING**

**Description**

Constructs sf of MULTILINESTRING objects, a helper for `sf_multilinestring()` with a simpler syntax.
Usage

sf_mline(obj, keep = FALSE, list_columns = NULL)

Arguments

- **obj**
  - sorted matrix or data.frame

- **keep**
  - logical indicating if the non-geometry and non-id columns should be kept. if TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.

- **list_columns**
  - vector of column names to turn into a list.

Value

- sf object of MULTILINESTRING geometries

Helpers

These are simpler versions of the main functions sf_point(), sf_multipoint(), sf_linestring(), sf_multilinestring(), sf_polygon(), and sf_multipolygon() for input data frame or matrix that contains columns appropriately of 'x', 'y', 'z', 'm', 'multipolygon_id', polygon_id', 'multilinestring_id', 'linestring_id', 'multipoint_id'.

This puts the onus of the naming and identification of entities onto the input data set, rather than when calling the creator function. This has pros and cons, so is not necessarily always 'simpler'. Please choose the appropriate constructor for the context you have. For examples a data frame from the real world with columns 'lon', 'lat', 'line' will be best used with

sf_linestring(df,x = "lon",y = "lat",linestring_id = "line")

whereas a heavy user of sfheaders might always create a data frame with 'x', 'y', 'linestring_id' precisely because they are expecting to call sf_line(df) and no further work is required. These are very different contexts and both equally valid.

Some columns are mandatory, such as 'x' and 'y' (always), while others depend on the output type where each column for that type is mandatory. The 'z' and/or 'm' values are included for 'XYZ', 'XYM', or 'XYZM' geometry types if and as they are present.

In summary these helpers:

- do not require arguments declaring column names.
- use assumed default column names, with no variation or absence allowed for a given type.
- use z, and/or m if present.
- use close = FALSE and keep = FALSE same as proper constructors.
- unlike sf_point() sf_pt() does not accept a flat vector for a single point.
- require a matrix or data frame with complete column names.

None of the helpers allow partial name matching for column names.
notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Keeping Properties

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

```r
m <- cbind(x = 0, y = 0, multilinestring_id = c(1, 1, 1), linestring_id = 1)
sf_mline(m)

df <- data.frame(
  multilinestring_id = c(1,1,1,1,1,1,2,2,2,2,2,2),
  linestring_id = c(1,1,1,2,2,3,3,3,1,1,1,2,2),
  x = rnorm(13),
  y = rnorm(13),
  z = rnorm(13),
  m = rnorm(13)
)
sf_mline(obj = df)
sf_mline(obj = df[-6])
## this gives XYZ, not XYM see #64
(sfx <- sf_mline(obj = df[-5]))

## we trivially round-trip with sf_mline()
sf_mline(sf_to_df(sfx))

## to round-trip with all fields use 'fill', then 'keep'
sf_mline(sf_to_df(sfx, fill = TRUE), keep = TRUE)
```

---

sf_mpoly

**Helper for sf MULTIPOLYGON**

**Description**

Constructs sf of MULTIPOLYGON objects, a helper for sf_multipolygon() with a simpler syntax.
Usage

sf_mpoly(obj, close = TRUE, keep = FALSE, list_columns = NULL)

Arguments

obj        sorted matrix or data.frame
close      logical indicating whether polygons should be closed. If TRUE, all polygons will
           be checked and force closed if possible
keep       logical indicating if the non-geometry and non-id columns should be kept. if
           TRUE you must supply the geometry and id columns, and only the first row of
           each geometry is kept. See Keeping Properties.
list_columns vector of column names to turn into a list.

Value

sf object of MULTIPOLYGON geometries

Helpers

These are simpler versions of the main functions sf_point(), sf_multipoint(), sf_linestring(),
sf_multilinestring(), sf_polygon(), and sf_multipolygon() for input data frame or matrix
that contains columns appropriately of 'x', 'y', 'z', 'm', 'multipolygon_id', polygon_id', 'multi-
linestring_id', 'linestring_id', 'multipoint_id'.

This puts the onus of the naming and identification of entities onto the input data set, rather than
when calling the creator function. This has pros and cons, so is not necessarily always 'simpler'.
Please choose the appropriate constructor for the context you have. For examples a data frame from
the real world with columns 'lon', 'lat', 'line' will be best used with

sf_linestring(df,x = "lon",y = "lat",linestring_id = "line")

whereas a heavy user of sfheaders might always create a data frame with 'x', 'y', 'linestring_id'
precisely because they are expecting to call sf_line(df) and no further work is required. These
are very different contexts and both equally valid.

Some columns are mandatory, such as 'x' and 'y' (always), while others depend on the output type
where each column for that type is mandatory. The 'z' and/or 'm' values are included for 'XYZ',
'XYM', or 'XYZM' geometry types if and as they are present.

In summary these helpers:

- do not require arguments declaring column names.
- use assumed default column names, with no variation or absence allowed for a given type.
- use z, and/or m if present.
- use close = FALSE and keep = FALSE same as proper constructors.
- unlike sf_point() sf_pt() does not accept a flat vector for a single point.
- require a matrix or data frame with complete column names.

None of the helpers allow partial name matching for column names.
notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Keeping Properties

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

```r
m <- matrix(c(0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0), ncol = 3, byrow = TRUE,
            dimnames = list(NULL, c("x", "y", "z")))
m <- cbind(m, multipolygon_id = 1, polygon_id = 1, linestring_id = 1)
sf_mpoly( m )

df <- as.data.frame(m)
sf_mpoly( df )
## order doesn't matter, only the names are used
sf_mpoly(df[c(6, 5, 3, 4, 1, 2)])
```

---

**sf_mpt**

*Helper for sf MULTIPOINT*

**Description**

Constructs sf of MULTIPOINT objects, a helper for `sf_multipoint()` with a simpler syntax.

**Usage**

```r
sf_mpt(obj, keep = FALSE, list_columns = NULL)
```
Arguments

- **obj**: sorted vector, matrix or data.frame
- **keep**: logical indicating if the non-geometry and non-id columns should be kept. If TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.
- **list_columns**: vector of column names to turn into a list.

Value

- sf object of MULTIPOINT geometries

Helpers

These are simpler versions of the main functions `sf_point()`, `sf_multipoint()`, `sf_linestring()`, `sf_multilinestring()`, `sf_polygon()`, and `sf_multipolygon()` for input data frame or matrix that contains columns appropriately of `x`, `y`, `z`, `m`, `multipolygon_id`, `polygon_id`, `multilinestring_id`, `linestring_id`, `multipoint_id`.

This puts the onus of the naming and identification of entities onto the input data set, rather than when calling the creator function. This has pros and cons, so is not necessarily always 'simpler'. Please choose the appropriate constructor for the context you have. For examples a data frame from the real world with columns 'lon', 'lat', 'line' will be best used with

```r
sf_linestring(df, x = "lon", y = "lat", linestring_id = "line")
```

whereas a heavy user of sfheaders might always create a data frame with 'x', 'y', 'linestring_id' precisely because they are expecting to call `sf_line(df)` and no further work is required. These are very different contexts and both equally valid.

Some columns are mandatory, such as 'x' and 'y' (always), while others depend on the output type where each column for that type is mandatory. The 'z' and/or 'm' values are included for 'XYZ', 'XYM', or 'XYZM' geometry types if and as they are present.

In summary these helpers:

- do not require arguments declaring column names.
- use assumed default column names, with no variation or absence allowed for a given type.
- use z, and/or m if present.
- use `close = FALSE` and `keep = FALSE` same as proper constructors.
- unlike `sf_point()` `sf_pt()` does not accept a flat vector for a single point.
- require a matrix or data frame with complete column names.

None of the helpers allow partial name matching for column names.

Notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.
Keeping Properties

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

```r
x <- cbind(x = 1:2, y = 3:4, multipoint_id = 1, ncol = 2 )
sf_mpt( x )

x <- data.frame( id = 1:2, x = 1:2, y = 2:1, multipoint_id = 1)
sf_mpt( x )
sf_mpt( x, keep = TRUE)

x <- data.frame(multipoint_id = 1:2, id = 1:2, x = 1:2, y = 2:1 )
sfx <- sf_mpt(x)

## we trivially round-trip with sf_mpt()
.sf_mpt(sf_to_df(sfx))
```

---

**sf_multilinestring**

**sf MULTILINESTRING**

**Description**

constructs an sf of MULTILINESTRING objects

**Usage**

```r
sf_multilinestring(
  obj = NULL,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  multilinestring_id = NULL,
  linestring_id = NULL,
  keep = FALSE,
  list_columns = NULL
)
```
Arguments

- **obj** sorted matrix or data.frame
- **x** x geometry column
- **y** y geometry column
- **z** z geometry column
- **m** m geometry column
- **multilinestring_id** column of ids for multilinestrings
- **linestring_id** column of ids for linestrings (within multilinestrings)
- **keep** logical indicating if the non-geometry and non-id columns should be kept. If TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.
- **list_columns** vector of column names to turn into a list.

Value

- sf object of MULTILINESTRING geometries

Notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Keeping Properties

Setting `keep = TRUE` will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input `obj`.

You can use `list_columns` to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in `list_columns`, only the first row of the column is kept.

The sf_* functions assume the input `obj` is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

```r
m <- matrix(c(0,0,0,0,1,1), ncol = 3 )
sf_multilinestring( m )

m <- matrix(c(0,0,0,0,0,1,0,1,1,2,3), ncol = 3, byrow = TRUE)
sf_multilinestring( obj = m )
sf_multilinestring( obj = m, multilinestring_id = 1 )
sf_multilinestring( obj = m, linestring_id = 1 )
sf_multilinestring( obj = m, linestring_id = 1, multilinestring_id = 1 )
```
sf_multipoint

**Description**

constructs sf of MULTIPOINT objects

**Usage**

```r
sf_multipoint(
  obj,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  multipoint_id = NULL,
  keep = FALSE,
  list_columns = NULL
)
```
Arguments

- **obj**: sorted matrix or data.frame
- **x**: x geometry column
- **y**: y geometry column
- **z**: z geometry column
- **m**: m geometry column
- **multipoint_id**: column of ids for multipoints
- **keep**: logical indicating if the non-geometry and non-id columns should be kept. If TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.
- **list_columns**: vector of column names to turn into a list.

Value

sf object of MULTIPOINT geometries

Notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Keeping Properties

Setting `keep = TRUE` will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input `obj`.

You can use `list_columns` to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in `list_columns`, only the first row of the column is kept.

The sf_* functions assume the input `obj` is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

```r
x <- matrix( c(1:4), ncol = 2 )
sf_multipoint( x )

x <- data.frame( id = 1:2, x = 1:2, y = 2:1 )
sf_multipoint( x )
sf_multipoint( x, x = "x", y = "y" )
sf_multipoint( x, x = "y", y = "x" )
sf_multipoint( x, multipoint_id = "id", x = "x", y = "y" )
```
sf_multipolygon

Description

constructs an sf of MULTIPOLYGON objects

Usage

sf_multipolygon(
  obj = NULL,
  x = NULL,
  y = NULL,
  z = NULL,
  m = NULL,
  multipolygon_id = NULL,
  polygon_id = NULL,
  linestring_id = NULL,
  close = TRUE,
  keep = FALSE,
  list_columns = NULL
)

Arguments

obj  sorted matrix or data.frame
x    x geometry column
y    y geometry column
z    z geometry column
m    m geometry column
multipolygon_id column of ids for multipolygons
polygon_id    column of ids for polygons
linestring_id column of ids for lines (within polygons)
close logical indicating whether polygons should be closed. If TRUE, all polygons will be checked and force closed if possible
keep logical indicating if the non-geometry and non-id columns should be kept. if TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.
list_columns vector of column names to turn into a list.

Value

sf object of MULTIPOLYGON geometries
notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Keeping Properties

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

m <- matrix(c(0,0,0,1,0,0,1,1,0,0,1,0,0,1,0,0), ncol = 3, byrow = TRUE )
sf_multipolygon( m )

df <- data.frame(
  id = c(1,1,1,1,1),
  x = c(0,0,1,1,0),
  y = c(0,1,1,0,0)
)
sf_multipolygon( df, x = "x", y = "y" )

df <- data.frame(
  id = c(1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2),
  x = c(0,0,1,0,1,1,2,2,2,2,2,2,2,2,2,2),
  y = c(0,1,1,0,0,1,2,2,2,2,2,2,2,2,2,2)
)
sf_multipolygon( df, multipolygon_id = "id", polygon_id = "id", linestring_id = "id")

df <- data.frame(
  id1 = c(1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1),
  id2 = c(1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2),
  x = c(0,0,1,0,1,1,2,2,2,2,2,2,2,2,2,2),
  y = c(0,1,1,0,0,1,2,2,2,2,2,2,2,2,2,2)
)
sf_multipolygon( df, multipolygon_id = "id1", polygon_id = "id2")

df <- data.frame(
  id1 = c(1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2),
  id2 = c(1,1,1,1,2,2,2,2,2,2,2,2,2,2,1,1),
  x = c(0,0,1,0,1,1,2,2,2,2,1,2,3,4,4,3)
)
Description

constructs sf of POINT objects

Usage

sf_point(obj, x = NULL, y = NULL, z = NULL, m = NULL, keep = FALSE)

Arguments

obj sorted vector, matrix or data.frame
x x geometry column
y y geometry column
z z geometry column
m m geometry column
keep logical indicating if the non-geometry and non-id columns should be kept. if TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.
Value

sf object of POINT geometries

Keeping Properties

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Examples

```r
x <- c(1:3)
sf_point(x)

x <- matrix(c(1:10), ncol = 2)
sf_point(x)

x <- setNames(as.data.frame(x), c("x","y"))
sf_point(x)
sf_point(obj = x, x = "x", y = "y")
sf_point(obj = x, x = "y", y = "x")

# keeping properties
x$val <- letters[1:5]
sf_point(x, x = "x", y = "y", keep = TRUE)
```

---

**sf_poly**

*Helper for sf POLYGON*

Description

Constructs sf of POLYGON objects, a helper for sf_polygon() with a simpler syntax.

Usage

```r
sf_poly(obj, close = TRUE, keep = FALSE, list_columns = NULL)
```
Arguments

obj  sorted matrix or data.frame

close  logical indicating whether polygons should be closed. If TRUE, all polygons will be checked and force closed if possible

keep  logical indicating if the non-geometry and non-id columns should be kept. if TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.

list_columns  vector of column names to turn into a list.

Value

sf  object of POLYGON geometries

Helpers

These are simpler versions of the main functions sf_point(), sf_multipoint(), sf_linestring(), sf_multilinestring(), sf_polygon(), and sf_multipolygon() for input data frame or matrix that contains columns appropriately of ‘x’, ‘y’, ‘z’, ‘m’, ‘multipolygon_id’, polygon_id’, ‘multilinestring_id’, ‘linestring_id’, ‘multipoint_id’.

This puts the onus of the naming and identification of entities onto the input data set, rather than when calling the creator function. This has pros and cons, so is not necessarily always ‘simpler’. Please choose the appropriate constructor for the context you have. For examples a data frame from the real world with columns ‘lon’, ‘lat’, ‘line’ will be best used with

\[
\text{sf_linestring(df, x = "lon", y = "lat", linestring_id = "line")}
\]

whereas a heavy user of sfheaders might always create a data frame with ‘x’, ‘y’, ‘linestring_id’ precisely because they are expecting to call sf_line(df) and no further work is required. These are very different contexts and both equally valid.

Some columns are mandatory, such as ‘x’ and ‘y’ (always), while others depend on the output type where each column for that type is mandatory. The ‘z’ and/or ‘m’ values are included for ‘XYZ’, ‘XYM’, or ‘XYZM’ geometry types if and as they are present.

In summary these helpers:

• do not require arguments declaring column names.
• use assumed default column names, with no variation or absence allowed for a given type.
• use z, and/or m if present.
• use close = FALSE and keep = FALSE same as proper constructors.
• unlike sf_point() sf_pt() does not accept a flat vector for a single point.
• require a matrix or data frame with complete column names.

None of the helpers allow partial name matching for column names.

notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.
**Keeping Properties**

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

**Examples**

```
m <- matrix(c(0,0,0,1,0,0,1,0,0,1,0,0,1,0,0,0), ncol = 3, byrow = TRUE,
   dimnames = list(NULL, c("x", "y", "z")))
m <- cbind(m, polygon_id = 1, linestring_id = 1)
sf_poly( m )

df <- as.data.frame(m)
sf_poly( df )

## order doesn’t matter, only the names are used
sf_poly(df[c(5, 3, 4, 1, 2)])
```

---

**sf_polygon**

**sf POLYGON**

**Description**

constructs an sf of POLYGON objects

**Usage**

```
sf_polygon(
   obj = NULL,
   x = NULL,
   y = NULL,
   z = NULL,
   m = NULL,
   polygon_id = NULL,
   linestring_id = NULL,
   close = TRUE,
   keep = FALSE,
   list_columns = NULL
)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>sorted matrix or data.frame</td>
</tr>
<tr>
<td>x</td>
<td>x geometry column</td>
</tr>
<tr>
<td>y</td>
<td>y geometry column</td>
</tr>
<tr>
<td>z</td>
<td>z geometry column</td>
</tr>
<tr>
<td>m</td>
<td>m geometry column</td>
</tr>
<tr>
<td>polygon_id</td>
<td>column of ids for polygons</td>
</tr>
<tr>
<td>linestring_id</td>
<td>column of ids for lines (within polygons)</td>
</tr>
<tr>
<td>close</td>
<td>logical indicating whether polygons should be closed. If TRUE, all polygons will be checked and force closed if possible</td>
</tr>
<tr>
<td>keep</td>
<td>logical indicating if the non-geometry and non-id columns should be kept. if TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.</td>
</tr>
<tr>
<td>list_columns</td>
<td>vector of column names to turn into a list.</td>
</tr>
</tbody>
</table>

Value

sf object of POLYGON geometries

Notes

sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.

The data.frame and matrices you send into the sfheader functions must be ordered.

Keeping Properties

Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.

You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept

The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

m <- matrix(c(0,0,0,0,1,1), ncol = 2 )
sf_polygon( m )

m <- matrix(c(0,0,0,0,1,1,2,2,1,3,3,4), ncol = 3, byrow = TRUE)
sf_polygon( obj = m )
sf_polygon( obj = m, polygon_id = 1 )
sf_polygon( obj = m, linestring_id = 1 )
sf_polygon( obj = m, linestring_id = 1, polygon_id = 1 )

sf_polygon( obj = m, x = 2, y = 3 )
sf_polygon( obj = m, x = 1, y = 2, z = 3 )
sf_polygon( obj = m, x = 2, y = 3, linestring_id = 1, polygon_id = 1 )

df <- data.frame(
    ml_id = c(1,1,1,1,1,1,1,2,2,2,2,2,2),
    l_id = c(1,1,2,2,2,3,3,3,1,1,2,2,2),
    x = rnorm(15),
    y = rnorm(15),
    z = rnorm(15),
    m = rnorm(15)
)
sf_polygon( obj = df, x = "x", y = "y")
sf_polygon( obj = df, x = "x", y = "y", z = "z")
sf_polygon( obj = df, x = "x", y = "y", z = "z", m = "m")
sf_polygon( obj = df, x = 2, y = 3)
sf_polygon( obj = df, x = 2, y = 3, z = 4)
sf_polygon( obj = df, x = 2, y = 3, z = 4, m = 5)
sf_polygon( obj = df, polygon_id = "ml_id", linestring_id = "l_id" )
sf_polygon( obj = df, polygon_id = 1, linestring_id = 2 )

## keeping properties
df <- data.frame(
    ml_id = c(1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2),
    l_id = c(1,1,2,2,2,3,3,3,1,1,2,2,2),
    x = rnorm(15),
    y = rnorm(15),
    z = rnorm(15),
    m = rnorm(15),
    val = letters[1:15],
    stringsAsFactors = FALSE
)

## using keep = TRUE means the first row of all non-geometries are kept
sf_polygon(
    obj = df,
    polygon_id = "ml_id",
    linestring_id = "l_id",
    x = "x",
    y = "y",
    keep = TRUE
)

## use 'list_column' to specify columns where you want to keep all the values
sf_polygon(
    obj = df,
    polygon_id = "ml_id",
    linestring_id = "l_id"
)
sf_pt

Helper for sf POINT

Description

Constructs sf of POINT objects, a helper for sf_point() with a simpler syntax.

Usage

sf_pt(obj, keep = FALSE)

Arguments

obj sorted vector, matrix or data.frame

keep logical indicating if the non-geometry and non-id columns should be kept. if TRUE you must supply the geometry and id columns, and only the first row of each geometry is kept. See Keeping Properties.

Value

sf object of POINT geometries

Helpers

These are simpler versions of the main functions sf_point(), sf_multipoint(), sf_linestring(), sf_multilinestring(), sf_polygon(), and sf_multipolygon() for input data frame or matrix that contains columns appropriately of 'x', 'y', 'z', 'm', 'multipolygon_id', polygon_id', 'multilinestring_id', 'linestring_id', 'multipoint_id'.

This puts the onus of the naming and identification of entities onto the input data set, rather than when calling the creator function. This has pros and cons, so is not necessarily always 'simpler'. Please choose the appropriate constructor for the context you have. For examples a data frame from the real world with columns 'lon', 'lat', 'line' will be best used with

sf_linestring(df,x = "lon",y = "lat",linestring_id = "line")

whereas a heavy user of sfheaders might always create a data frame with 'x', 'y', 'linestring_id' precisely because they are expecting to call sf_line(df) and no further work is required. These are very different contexts and both equally valid.

Some columns are mandatory, such as 'x' and 'y' (always), while others depend on the output type where each column for that type is mandatory. The 'z' and/or 'm' values are included for 'XYZ', 'XYM', or 'XYZM' geometry types if and as they are present.

In summary these helpers:
• do not require arguments declaring column names.
• use assumed default column names, with no variation or absence allowed for a given type.
• use z, and/or m if present.
• use close = FALSE and keep = FALSE same as proper constructors.
• unlike sf_point() sf_pt() does not accept a flat vector for a single point.
• require a matrix or data frame with complete column names.

None of the helpers allow partial name matching for column names.

notes
sfheaders functions do not perform any validity checks on the geometries. Nor do they set Coordinate Reference Systems, EPSG, PROJ4 or precision attributes.
The data.frame and matrices you send into the sfheader functions must be ordered.

Keeping Properties
Setting keep = TRUE will retain any columns not specified as a coordinate (x, y, z, m) or an id (e.g., linestring_id, polygon_id) of the input obj.
You can use list_columns to specify which of the properties will be turned into a list, thus keeping all the values in the column. For columns not specified in list_columns, only the first row of the column is kept.
The sf_* functions assume the input obj is a long data.frame / matrix, where any properties are repeated down the table for the same geometry.

Examples

```
x <- cbind(x = 1, y = 3)
sf_pt( x )
sf_pt(cbind(x, z = 2))

x <- matrix( c(1:10) , ncol = 2 , dimnames = list(NULL, c("x", "y")))
sf_pt( x )

x <- setNames( as.data.frame( x ), c("x","y") )
sf_pt( x )

# keeping properties
x$val <- letters[1:5]
(sfx <- sf_pt( x, keep = TRUE ))

## we trivially round-trip with sf_pt()
sf_pt(sf_to_df(sfx, fill = TRUE), keep = TRUE)
```
Description

Removes holes from polygons and multipolygons. Points and linestrings are unaffected.

Usage

sf_remove_holes(obj, close = TRUE)

Arguments

obj sfg, sfc or sf object.

close logical indicating whether polygons should be closed. If TRUE, all polygons will be checked and force closed if possible

Examples

df <- data.frame(
  ml_id = c(1,1,1,1,1,1,1,1,2,2,2,2,2,2,2),
  l_id = c(1,1,1,2,2,2,3,3,3,1,1,1,2,2,2),
  x = rnorm(15),
  y = rnorm(15),
  z = rnorm(15),
  m = rnorm(15)
)

sfg <- sfg_polygon( obj = df, x = "x", y = "y", linestring_id = "ml_id" )
sfc <- sfc_polygon( obj = df, x = "x", y = "y", polygon_id = "ml_id", linestring_id = "l_id" )
sf <- sf_polygon( obj = df, x = "x", y = "y", polygon_id = "ml_id", linestring_id = "l_id" )

sf_remove_holes( sfg )
sf_remove_holes( sfc )
sf_remove_holes( sf )

Description

Converts an sf object to a data.frame

Usage

sf_to_df(sf, fill = FALSE, unlist = NULL)
sf_to_df

Arguments

sf sf object
fill logical indicating if the resulting data.frame should be filled with the data columns from the sf object. If TRUE, each row of data will be replicated for every coordinate in every geometry.
unlist string vector of columns to unlist. Each list element is equivalent to a row of the input object, and is expected to be the same length as the number of coordinates in the geometry.

Examples

df <- data.frame(
m_id = c(1,1,1,1,1,1,1,1,1,2,2,2,2,2,2)
, l_id = c(1,1,1,2,2,2,3,3,3,1,1,2,2,2)
, x = rnorm(15)
, y = rnorm(15)
, z = rnorm(15)
, m = rnorm(15)
)
sf <- sf_polygon( obj = df, polygon_id = "m_id", linestring_id = "l_id" )
df <- sf_to_df( sf )
## with associated data
sf$val1 <- c("a","b")
sf$val2 <- c(1L, 2L)
df <- sf_to_df( sf, fill = TRUE )
## Unlisting list columns

df <- data.frame(
l_id = c(1,1,1,2,2,3,3,3,3,3)
, x = rnorm(10)
, y = rnorm(10)
)
sf <- sf_linestring( obj = df, linestring_id = "l_id" , x = "x" , y = "y" )
## put on a list column
sf$l <- list( c(1,2,3),c(3,2,1),c(10,11,12,13))
sf_to_df( sf, unlist = "l" )
Index

sf_bbox, 21
sf_boxes, 23
sf_cast, 24
sf_line, 25
sf_linestring, 27
sf_linestring(), 25, 29, 31, 33, 42, 46
sf_mline, 28
sf_mpoly, 30
sf_mpt, 32
sf_multilinestring, 34
sf_multilinestring(), 25, 28, 29, 31, 33, 42, 46
sf_multipoint, 36
sf_multipoint(), 25, 29, 31–33, 42, 46
sf_multipolygon, 38
sf_multipolygon(), 25, 29–31, 33, 42, 46
sf_point, 40
sf_point(), 25, 26, 29, 31, 33, 42, 46, 47
sf_poly, 41
sf_polygon, 43
sf_polygon(), 25, 29, 31, 33, 41, 42, 46
sf_pt, 46
sf_pt(), 26, 29, 31, 33, 42, 47
sf_remove_holes, 48
sf_to_df, 48
sfc_cast, 2
sfc_linestring, 3
sfc_multilinestring, 5
sfc_multipoint, 6
sfc_multipolygon, 8
sfc_point, 10
sfc_polygon, 11
sfc_to_df, 13
sfg_linestring, 13
sfg_multilinestring, 14
sfg_multipoint, 15
sfg_multipolygon, 16
sfg_point, 18
sfg_polygon, 19

sfg_to_df, 20