

Package: sdf (via r-universe)

October 16, 2026

Title An Extensible Spatial Data Frame

Version 0.1.0

Description A spatial data frame that is agnostic to the geometry representation it holds. A geometry vector becomes usable by implementing a small set of generics, in the manner of a trait, after which the spatial data frame, its 'dplyr' methods, and the spatial join and filter work unchanged. Implementations for the 'sf', 'geos', and 's2' packages are included.

License MIT + file LICENSE

URL <https://github.com/JosiahParry/sdf>

BugReports <https://github.com/JosiahParry/sdf/issues>

Suggests duckspatial, geos, s2, sedonadb, sf, testthat (>= 3.0.0),
usethis

Config/testthat/edition 3

Encoding UTF-8

Language en

Roxygen list(markdown = TRUE)

Imports cli, dplyr, rlang, stats, tibble, tidyselect, vctrs, wk

Config/roxygen2/version 8.1.0

Repository <https://josiahparry.r-universe.dev>

Date/Publication 2026-09-13 21:24:00 UTC

RemoteUrl <https://github.com/josiahparry/sdf>

RemoteRef HEAD

RemoteSha 7e0789674478af09f22119e33f2e3e694a8ece26

Contents

[.sdf	2
as_sdf	3

is_geometry	4
sdf_geometry	5
st_distance	5
st_geometry_type	6
st_intersection	7
st_intersects	8
st_is_valid	9
st_join	10
st_length	12
st_make_valid	12
st_relate	13
st_union	14
use_sdf_template_class	15
Index	16

[.sdf	<i>Subset a spatial data frame</i>
-------	------------------------------------

Description

Keeps the geometry column and the sdf class through a [subset, which the tibble method alone would drop.

Usage

```
## S3 method for class 'sdf'
x[i, j, ...]
```

Arguments

- x a spatial data frame.
- i, j rows and columns to keep.
- ... passed on to the tibble method.

Value

A spatial data frame.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

  x <- as_sdf(tibble::tibble(
    name = as.character(nc$NAME),
    geometry = sf::st_geometry(nc)
  ))
}
```

```
x[1:3, ]
}
```

as_sdf

Create a spatial data frame

Description

as_sdf() promotes a data frame that already holds a geometry column. new_sdf() builds one from a data frame and a geometry vector.

Usage

```
as_sdf(x)
```

```
new_sdf(x, geometry)
```

Arguments

x for as_sdf(), a data frame holding a geometry column. For new_sdf(), a data frame or named list of the other columns.

geometry a geometry vector, as long as x has rows.

Details

The geometry column is found with [is_geometry\(\)](#), so any vector whose class implements the sdf generics can be the geometry. The bounding box is computed once with [st_extent\(\)](#) and carried as an attribute.

Value

A spatial data frame, a tibble of class sdf.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

  as_sdf(tibble::tibble(
    name = as.character(nc$NAME),
    geometry = sf::st_geometry(nc)
  ))

  new_sdf(list(name = as.character(nc$NAME)), sf::st_geometry(nc))
}
```

`is_geometry`*Required generics*

Description

The three generics a geometry vector must implement to work as the geometry column of a spatial data frame.

Usage

```
is_geometry(x)
```

```
st_extent(x)
```

```
st_collect(x)
```

Arguments

`x` a geometry vector.

Details

`is_geometry()` must return TRUE for the class. `st_extent()` must return a numeric vector named `xmin`, `ymin`, `xmax`, `ymax` covering the whole vector. `st_collect()` must return the vector as a single geometry, following PostGIS ST_Collect: a MULTI* where the parts agree on a type and a GEOMETRYCOLLECTION where they do not. The defaults use wk, so a class with a wk handler already has all three.

Value

`is_geometry()` a single logical. `st_extent()` a named numeric vector of length 4. `st_collect()` a geometry vector of length 1.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {  
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)  
  geometry <- sf::st_geometry(nc)  
  
  is_geometry(geometry)  
  st_extent(geometry)  
  st_collect(geometry)  
}
```

sdf_geometry	<i>Geometry accessor</i>
--------------	--------------------------

Description

Pulls the geometry column out of a spatial data frame, whatever it is named.

Usage

```
sdf_geometry(x)
```

Arguments

`x` a spatial data frame.

Value

The geometry vector of `x`.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {  
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)  
  x <- as_sdf(tibble::tibble(  
    name = as.character(nc$NAME),  
    geometry = sf::st_geometry(nc)  
  ))  
  
  sdf_geometry(x)  
}
```

st_distance	<i>Measurement</i>
-------------	--------------------

Description

ST_Distance between two vectors and ST_Perimeter of one, both returning plain numerics as PostGIS returns floats.

Usage

```
st_distance(x, y, ...)
```

```
st_perimeter(x, ...)
```

Arguments

x, y geometry vectors.
 ... passed to the method.

Value

A numeric vector.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geometry <- sf::st_geometry(nc)

  st_distance(geometry[1:3], geometry[4:6])
  st_perimeter(geometry[1:3])
}
```

st_geometry_type	<i>Read parts of each geometry</i>
------------------	------------------------------------

Description

ST_GeometryType, ST_Envelope, ST_Boundary, ST_NumGeometries, ST_NumPoints, ST_X and ST_Y, one answer per geometry.

Usage

```
st_geometry_type(x, ...)

st_envelope(x, ...)

st_boundary(x, ...)

st_num_geometries(x, ...)

st_num_points(x, ...)

st_x(x, ...)

st_y(x, ...)
```

Arguments

x a geometry vector.
 ... passed to the method.

Details

st_envelope() is the bounding box of each geometry, unlike [st_extent\(\)](#), which is the one box covering the whole vector. st_x() and st_y() are defined for points and give NA elsewhere.

Value

st_geometry_type() a character vector. st_envelope() and st_boundary() a geometry vector. The rest a numeric vector. All as long as x.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geometry <- sf::st_geometry(nc)[1:3]

  st_geometry_type(geometry)
  st_boundary(geometry)
}

# sf has no per geometry envelope, geometry count, point count, x or y, so
# those are answered by the backends that do
if (requireNamespace("geos", quietly = TRUE)) {
  pts <- geos::as_geos_geometry(c("POINT (0 1)", "POINT (2 3)"))

  st_envelope(pts)
  st_num_geometries(pts)
  st_x(pts)
  st_y(pts)
}
```

st_intersection	<i>Overlay two geometry vectors</i>
-----------------	-------------------------------------

Description

The pairwise overlay operations, ST_Intersection, ST_Difference and ST_SymDifference.

Usage

```
st_intersection(x, y, ...)

st_difference(x, y, ...)

st_sym_difference(x, y, ...)
```

Arguments

x, y	geometry vectors.
...	passed to the method.

Details

These pair the two vectors element by element, unlike the predicates, which compare every row of x against every row of y.

Value

A geometry vector.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  a <- sf::st_geometry(nc)[1:3]
  b <- sf::st_buffer(a, 0.05)

  st_intersection(a, b)
  st_difference(b, a)
}
```

st_intersects

Spatial predicates

Description

Each returns, for every row of x, the row positions of y that the relationship holds for. Implementing them is optional, but they are what [st_join\(\)](#) and [st_filter\(\)](#) run on.

Usage

```
st_intersects(x, y, ...)

st_contains(x, y, ...)

st_within(x, y, ...)

st_crosses(x, y, ...)

st_covers(x, y, ...)

st_covered_by(x, y, ...)

st_equals(x, y, ...)

st_disjoint(x, y, ...)

st_touches(x, y, ...)
```

```
st_overlaps(x, y, ...)

st_dwithin(x, y, distance, ...)
```

Arguments

x, y	geometry vectors.
...	passed to the method.
distance	the distance within which the relationship holds.

Details

A method returns a list as long as x, whose elements are the row positions of y. Integer is the natural type but doubles are accepted, and an element of length zero means no match. An sf sgbp already has this shape.

Value

A list as long as x, each element the row positions of y that match.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geometry <- sf::st_geometry(nc)

  st_intersects(geometry[1:3], geometry[1:10])
}
```

st_is_valid	<i>Test each geometry</i>
-------------	---------------------------

Description

ST_IsValid, ST_IsEmpty and ST_IsSimple, one answer per geometry.

Usage

```
st_is_valid(x, ...)

st_is_empty(x, ...)

st_is_simple(x, ...)
```

Arguments

x	a geometry vector.
...	passed to the method.

Value

A logical vector as long as `x`.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geometry <- sf::st_geometry(nc)[1:3]

  st_is_valid(geometry)
  st_is_empty(geometry)
}
```

st_join

Spatial join and filter

Description

`st_join()` attaches the columns of `y` to each row of `x` that it relates to spatially. `st_filter()` keeps the rows of `x` that relate to any row of `y`, without adding columns.

Usage

```
st_join(
  x,
  y,
  predicate = st_intersects,
  ...,
  join_type = c("left", "inner"),
  suffix = c("_x", "_y")
)

## Default S3 method:
st_join(
  x,
  y,
  predicate = st_intersects,
  ...,
  join_type = c("left", "inner"),
  suffix = c("_x", "_y")
)

st_filter(x, y, predicate = st_intersects, ...)
```

Default S3 method:

```
st_filter(x, y, predicate = st_intersects, ...)
```

Arguments

x, y	spatial data frames.
predicate	a function of x and y returning the row positions of y matching each row of x, such as <code>st_intersects()</code> .
...	passed to predicate.
join_type	"left" to keep the rows of x that match nothing, "inner" to drop them.
suffix	a character vector of length 2, the suffixes for names the two sides share.

Details

Both dispatch on x, so a backend that holds its data somewhere other than R, such as a database, can replace either and run the whole operation there. The default works for any backend whose geometry vector can be subset with `[]` and whose predicate returns row positions.

The default puts y into the geometry type of x before the predicate sees it, so the two sides need not hold the same type.

A row of x matching several rows of y is repeated once per match. A row matching nothing is kept against NA when join_type is "left" and dropped when it is "inner". Names the two sides share are suffixed, and the geometry of the result is the geometry of x.

Not every backend offers both. One that cannot do a left join says so rather than returning an inner join under its name.

Value

`st_join()` returns a spatial data frame of the matched rows. `st_filter()` returns x with the unmatched rows dropped.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

  counties <- as_sdf(tibble::tibble(
    name = as.character(nc$NAME),
    geometry = sf::st_geometry(nc)
  ))

  sites <- as_sdf(tibble::tibble(
    id = 1:2,
    geometry = sf::st_sfc(
      sf::st_point(c(-78.6, 35.8)),
      sf::st_point(c(-80.8, 35.2)),
      crs = sf::st_crs(nc)
    )
  ))

  st_join(sites, counties, predicate = st_within)
  st_filter(counties, sites)
}
```

st_length	<i>Optional generics</i>
-----------	--------------------------

Description

Common measures a backend may implement.

Usage

```
st_length(x, ...)
```

```
st_area(x, ...)
```

Arguments

x	a geometry vector.
...	passed to the method.

Value

A numeric vector as long as x.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)

  st_area(sf::st_geometry(nc))
}
```

st_make_valid	<i>Rebuild each geometry</i>
---------------	------------------------------

Description

ST_MakeValid, ST_Reverse and ST_PointOnSurface, one geometry out per geometry in.

Usage

```
st_make_valid(x, ...)
```

```
st_reverse(x, ...)
```

```
st_point_on_surface(x, ...)
```

Arguments

`x` a geometry vector.
`...` passed to the method.

Value

A geometry vector as long as `x`.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geometry <- sf::st_geometry(nc)[1:3]

  st_make_valid(geometry)
  st_point_on_surface(geometry)
}
```

st_relate

The DE-9IM relationship between two vectors

Description

ST_Relate, the pattern every other predicate is a special case of.

Usage

```
st_relate(x, y, ...)
```

Arguments

`x, y` geometry vectors.
`...` passed to the method.

Details

This pairs the two vectors element by element and returns the pattern for each pair, so it is not a predicate `st_join()` can run on.

Value

A character vector of DE-9IM patterns.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geometry <- sf::st_geometry(nc)

  st_relate(geometry[1:3], geometry[2:4])
}
```

st_union

*Primary interface***Description**

Optional but strongly suggested: the operations a spatial data frame is usually asked for.

Usage

```
st_union(x)

st_simplify(x, ...)

st_buffer(x, distance, ...)

st_centroid(x)

st_convex_hull(x)

st_concave_hull(x, concavity, ...)
```

Arguments

x	a geometry vector.
...	passed to the method.
distance	the buffer distance.
concavity	how tightly the hull follows the points.

Value

A geometry vector. st_union() returns one of length 1.

Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
  geometry <- sf::st_geometry(nc)

  st_centroid(geometry)
  st_convex_hull(geometry)
}
```

`use_sdf_template_class`*Templates to extend sdf*

Description

These templates can be used to extend sdf by implementing a new compatible geometry vector column.

Usage

```
use_sdf_template_class(  
  class,  
  save_as = paste0("R/sdf-compat-", class, ".R"),  
  open = TRUE  
)  
  
use_sdf_template_zzz(class, save_as = "R/zzz.R", open = TRUE)
```

Arguments

<code>class</code>	a scalar character of the class name of your geometry class
<code>save_as</code>	Path of file to create, relative to root of active project. Defaults to <code>template</code>
<code>open</code>	Open the newly created file for editing? Happens in RStudio, if applicable, or via <code>utils::file.edit()</code> otherwise.

Details

- `use_sdf_template_class()` will create an R script with the scaffolding needed to implement your geometry class to be compatible with sdf.
- `use_sdf_template_zzz()` will create a `zzz.R` file that will export your methods to be available to sdf.

Index

[.sdf, [2](#)

as_sdf, [3](#)

is_geometry, [4](#)
is_geometry(), [3](#)

new_sdf (as_sdf), [3](#)

sdf_geometry, [5](#)
st_area (st_length), [12](#)
st_boundary (st_geometry_type), [6](#)
st_buffer (st_union), [14](#)
st_centroid (st_union), [14](#)
st_collect (is_geometry), [4](#)
st_concave_hull (st_union), [14](#)
st_contains (st_intersects), [8](#)
st_convex_hull (st_union), [14](#)
st_covered_by (st_intersects), [8](#)
st_covers (st_intersects), [8](#)
st_crosses (st_intersects), [8](#)
st_difference (st_intersection), [7](#)
st_disjoint (st_intersects), [8](#)
st_distance, [5](#)
st_dwithin (st_intersects), [8](#)
st_envelope (st_geometry_type), [6](#)
st_equals (st_intersects), [8](#)
st_extent (is_geometry), [4](#)
st_extent(), [3](#), [7](#)
st_filter (st_join), [10](#)
st_filter(), [8](#)
st_geometry_type, [6](#)
st_intersection, [7](#)
st_intersects, [8](#)
st_intersects(), [11](#)
st_is_empty (st_is_valid), [9](#)
st_is_simple (st_is_valid), [9](#)
st_is_valid, [9](#)
st_join, [10](#)
st_join(), [8](#), [13](#)
st_length, [12](#)
st_make_valid, [12](#)
st_num_geometries (st_geometry_type), [6](#)
st_num_points (st_geometry_type), [6](#)
st_overlaps (st_intersects), [8](#)
st_perimeter (st_distance), [5](#)
st_point_on_surface (st_make_valid), [12](#)
st_relate, [13](#)
st_reverse (st_make_valid), [12](#)
st_simplify (st_union), [14](#)
st_sym_difference (st_intersection), [7](#)
st_touches (st_intersects), [8](#)
st_union, [14](#)
st_within (st_intersects), [8](#)
st_x (st_geometry_type), [6](#)
st_y (st_geometry_type), [6](#)
use_sdf_template_class, [15](#)
use_sdf_template_zzz
 (use_sdf_template_class), [15](#)
utils::file.edit(), [15](#)