Package 's2'

May 23, 2025

Title Spherical Geometry Operators Using the S2 Geometry Library

Version 1.1.9

Description Provides R bindings for Google's s2 library for geometric calculations on the sphere. High-performance constructors and exporters provide high compatibility with existing spatial packages, transformers construct new geometries from existing geometries, predicates provide a means to select geometries based on spatial relationships, and accessors extract information about geometries.

License Apache License (== 2.0)

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

SystemRequirements cmake, OpenSSL >= 1.0.1, Abseil >= 20230802.0

LinkingTo Rcpp, wk

Imports Rcpp, wk (>= 0.6.0)

Suggests bit64, testthat (>= 3.0.0), vctrs

URL https://r-spatial.github.io/s2/, https://github.com/r-spatial/s2, http://s2geometry.io/

BugReports https://github.com/r-spatial/s2/issues

Depends R (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation yes

Author Dewey Dunnington [aut] (ORCID: <https://orcid.org/0000-0002-9415-4582>),
Edzer Pebesma [aut, cre] (ORCID:
<https://orcid.org/0000-0001-8049-7069>),
Ege Rubak [aut],
Jeroen Ooms [ctb] (configure script),
Google, Inc. [cph] (Original s2geometry.io source code)

Maintainer Edzer Pebesma <edzer.pebesma@uni-muenster.de>

Repository CRAN

Date/Publication 2025-05-23 15:50:02 UTC

Contents

as_s2_geography . . . 2
s2_boundary . . . 4
s2_bounds_cap . . . 7
s2_cell . . . 8
s2_cell_is_valid . . . 10
s2_cell_union . . . 11
s2_cell_union_normalize . . . 12
s2_closest_feature . . . 13
s2_contains . . . 15
s2_data_example_wkt . . . 18
s2_data_tbl_countries . . . 19
s2_earth_radius_meters . . . 20
s2_geog_point . . . 20
s2_is_collection . . . 23
s2_lnglat . . . 25
s2_options . . . 26
s2_plot . . . 28
s2_point . . . 29
s2_project . . . 30
wk_handle.s2_geography . . . 31

Index **34**

`as_s2_geography` *Create an S2 Geography Vector*

Description

Geography vectors are arrays of points, lines, polygons, and/or collections of these. Geography vectors assume coordinates are longitude and latitude on a perfect sphere.

Usage

```
as_s2_geography(x, ...)

s2_geography()

## S3 method for class 's2_geography'
as_s2_geography(x, ...)

## S3 method for class 'wk_xy'
as_s2_geography(x, ...)

## S3 method for class 'wk_wkb'
as_s2_geography(x, ..., oriented = FALSE, check = TRUE)
```

```
## S3 method for class 'WKB'
as_s2_geography(x, ..., oriented = FALSE, check = TRUE)

## S3 method for class 'blob'
as_s2_geography(x, ..., oriented = FALSE, check = TRUE)

## S3 method for class 'wk_wkt'
as_s2_geography(x, ..., oriented = FALSE, check = TRUE)

## S3 method for class 'character'
as_s2_geography(x, ..., oriented = FALSE, check = TRUE)

## S3 method for class 'logical'
as_s2_geography(x, ...)

## S3 method for class 's2_geography'
as_wkb(x, ...)

## S3 method for class 's2_geography'
as_wkt(x, ...)
```

Arguments

x	An object that can be converted to an s2_geography vector
...	Unused
oriented	TRUE if polygon ring directions are known to be correct (i.e., exterior rings are defined counter clockwise and interior rings are defined clockwise).
check	Use check = FALSE to skip error on invalid geometries

Details

The coercion function as_s2_geography() is used to wrap the input of most functions in the s2 package so that you can use other objects with an unambiguious interpretation as a geography vector. Geography vectors have a minimal vctrs implementation, so you can use these objects in tibble, dplyr, and other packages that use the vctrs framework.

Value

An object with class s2_geography

See Also

s2_geog_from_wkb(), s2_geog_from_text(), s2_geog_point(), s2_make_line(), s2_make_polygon() for other ways to create geography vectors, and s2_as_binary() and s2_as_text() for other ways to export them.

s2_boundary *S2 Geography Transformations*

Description

These functions operate on one or more geography vectors and return a geography vector.

Usage

```
s2_boundary(x)

s2_centroid(x)

s2_closest_point(x, y)

s2_minimum_clearance_line_between(x, y)

s2_difference(x, y, options = s2_options())

s2_sym_difference(x, y, options = s2_options())

s2_intersection(x, y, options = s2_options())

s2_union(x, y = NULL, options = s2_options())

s2_snap_to_grid(x, grid_size)

s2_simplify(x, tolerance, radius = s2_earth_radius_meters())

s2_rebuild(x, options = s2_options())

s2_buffer_cells(
  x,
  distance,
  max_cells = 1000,
  min_level = -1,
  radius = s2_earth_radius_meters()
)

s2_convex_hull(x)

s2_centroid_agg(x, na.rm = FALSE)

s2_coverage_union_agg(x, options = s2_options(), na.rm = FALSE)

s2_rebuild_agg(x, options = s2_options(), na.rm = FALSE)
```

```
s2_union_agg(x, options = s2_options(), na.rm = FALSE)

s2_convex_hull_agg(x, na.rm = FALSE)

s2_point_on_surface(x, na.rm = FALSE)
```

Arguments

x, y	geography vectors. These inputs are passed to as_s2_geography(), so you can pass other objects (e.g., character vectors of well-known text) directly.
options	An s2_options() object describing the polygon/polyline model to use and the snap level.
grid_size	The grid size to which coordinates should be snapped; will be rounded to the nearest power of 10.
tolerance	The minimum distance between vertexes to use when simplifying a geography.
radius	Radius of the earth. Defaults to the average radius of the earth in meters as defined by s2_earth_radius_meters().
distance	The distance to buffer, in units of radius.
max_cells	The maximum number of cells to approximate a buffer.
min_level	The minimum cell level used to approximate a buffer (1 - 30). Setting this value too high will result in unnecessarily large geographies, but may help improve buffers along long, narrow regions.
na.rm	For aggregate calculations use na.rm = TRUE to drop missing values.

Model

The geometry model indicates whether or not a geometry includes its boundaries. Boundaries of line geometries are its end points. OPEN geometries do not contain their boundary (model = "open"); CLOSED geometries (model = "closed") contain their boundary; SEMI-OPEN geometries (model = "semi-open") contain half of their boundaries, such that when two polygons do not overlap or two lines do not cross, no point exist that belong to more than one of the geometries. (This latter form, half-closed, is not present in the OpenGIS "simple feature access" (SFA) standard nor DE9-IM on which that is based). The default values for s2_contains() (open) and covers/covered_by (closed) correspond to the SFA standard specification of these operators.

See Also

BigQuery's geography function reference:

- ST_BOUNDARY
- ST_CENTROID
- ST_CLOSESTPOINT
- ST_DIFFERENCE
- ST_INTERSECTION
- ST_UNION

- ST_SNAPTOGRID
- ST_SIMPLIFY
- ST_UNION_AGG
- ST_CENTROID_AGG

Examples

```
# returns the boundary:
# empty for point, endpoints of a linestring,
# perimeter of a polygon
s2_boundary("POINT (-64 45)")
s2_boundary("LINESTRING (0 0, 10 0)")
s2_boundary("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))")

# returns the area-weighted centroid, element-wise
s2_centroid("POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))")
s2_centroid("LINESTRING (0 0, 10 0)")

# s2_point_on_surface guarantees a point on surface
# Note: this is not the same as st_point_on_surface
s2_centroid("POLYGON ((0 0, 10 0, 1 1, 0 10, 0 0))")
s2_point_on_surface("POLYGON ((0 0, 10 0, 1 1, 0 10, 0 0))")

# returns the unweighted centroid of the entire input
s2_centroid_agg(c("POINT (0 0)", "POINT (10 0)"))

# returns the closest point on x to y
s2_closest_point(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  "POINT (0 90)" # north pole!
)

# returns the shortest possible line between x and y
s2_minimum_clearance_line_between(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  "POINT (0 90)" # north pole!
)

# binary operations: difference, symmetric difference, intersection and union
s2_difference(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  "POLYGON ((5 5, 15 5, 15 15, 5 15, 5 5))",
  # 32 bit platforms may need to set snap rounding
  s2_options(snap = s2_snap_level(30))
)

s2_sym_difference(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  "POLYGON ((5 5, 15 5, 15 15, 5 15, 5 5))",
  # 32 bit platforms may need to set snap rounding
  s2_options(snap = s2_snap_level(30))
)
```

```
s2_intersection(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  "POLYGON ((5 5, 15 5, 15 15, 5 15, 5 5))",
  # 32 bit platforms may need to set snap rounding
  s2_options(snap = s2_snap_level(30))
)

s2_union(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  "POLYGON ((5 5, 15 5, 15 15, 5 15, 5 5))",
  # 32 bit platforms may need to set snap rounding
  s2_options(snap = s2_snap_level(30))
)

# s2_convex_hull_agg builds the convex hull of a list of geometries
s2_convex_hull_agg(
  c(
    "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
    "POLYGON ((5 5, 15 5, 15 15, 5 15, 5 5))"
  )
)

# use s2_union_agg() to aggregate geographies in a vector
s2_coverage_union_agg(
  c(
    "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
    "POLYGON ((5 5, 15 5, 15 15, 5 15, 5 5))"
  ),
  # 32 bit platforms may need to set snap rounding
  s2_options(snap = s2_snap_level(30))
)

# snap to grid rounds coordinates to a specified grid size
s2_snap_to_grid("POINT (0.333333333333 0.666666666666)", 1e-2)
```

---

s2_bounds_cap *Compute feature-wise and aggregate bounds*

Description

s2_bounds_rect() returns a bounding latitude-longitude rectangle that contains the region; s2_bounds_cap() returns a bounding circle represented by a centre point (lat, lng) and an angle. The bound may not be tight for points, polylines and geometry collections. The rectangle returned may depend on the order of points or polylines. `lng_lo` values larger than `lng_hi` indicate regions that span the antimeridian, see the Fiji example.

Usage

```
s2_bounds_cap(x)

s2_bounds_rect(x)
```

Arguments

x	An s2_geography() vector.

Value

Both functions return a data.frame:

- s2_bounds_rect(): Columns minlng, minlat, maxlng, maxlat (degrees)
- s2_bounds_cap(): Columns lng, lat, angle (degrees)

Examples

```
s2_bounds_cap(s2_data_countries("Antarctica"))
s2_bounds_cap(s2_data_countries("Netherlands"))
s2_bounds_cap(s2_data_countries("Fiji"))

s2_bounds_rect(s2_data_countries("Antarctica"))
s2_bounds_rect(s2_data_countries("Netherlands"))
s2_bounds_rect(s2_data_countries("Fiji"))
```

---

s2_cell *Create S2 Cell vectors*

Description

The S2 cell indexing system forms the basis for spatial indexing in the S2 library. On their own, S2 cells can represent points or areas. As a union, a vector of S2 cells can approximate a line or polygon. These functions allow direct access to the S2 cell indexing system and are designed to have minimal overhead such that looping and recursion have acceptable performance when used within R code.

Usage

```
s2_cell(x = character())

s2_cell_sentinel()

s2_cell_invalid()

as_s2_cell(x, ...)
```

```
## S3 method for class 's2_cell'
as_s2_cell(x, ...)

## S3 method for class 'character'
as_s2_cell(x, ...)

## S3 method for class 's2_geography'
as_s2_cell(x, ...)

## S3 method for class 'wk_xy'
as_s2_cell(x, ...)

## S3 method for class 'integer64'
as_s2_cell(x, ...)

new_s2_cell(x)
```

Arguments

x	The canonical S2 cell identifier as a character vector.
...	Passed to methods

Details

Under the hood, S2 cell vectors are represented in R as vectors of type `double()`. This works because S2 cell identifiers are 64 bits wide, as are `double`s on all systems where R runs (The same trick is used by the bit64 package to represent signed 64-bit integers). As a happy accident, `NA_real_` is not a valid or meaningful cell identifier, so missing value support in the way R users might expect is preserved. It is worth noting that the underlying value of `s2_cell_sentinel()` would normally be considered `NA`; however, as it is meaningful and useful when programming with S2 cells, custom `is.na()` and comparison methods are implemented such that `s2_cell_sentinel()` is greater than all valid S2 cells and not considered missing. Users can and should implement compiled code that uses the underlying bytes of the vector, ensuring that the class of any returned object that should be interpreted in this way is constructed with `new_s2_cell()`.

Value

An object of class s2_cell

Examples

```
s2_cell("4b59a0cd83b5de49")
as_s2_cell(s2_lnglat(-64, 45))
as_s2_cell(s2_data_cities("Ottawa"))
```

s2_cell_is_valid *S2 cell operators*

Description

S2 cell operators

Usage

```
s2_cell_is_valid(x)

s2_cell_debug_string(x)

s2_cell_to_lnglat(x)

s2_cell_center(x)

s2_cell_boundary(x)

s2_cell_polygon(x)

s2_cell_vertex(x, k)

s2_cell_level(x)

s2_cell_is_leaf(x)

s2_cell_is_face(x)

s2_cell_area(x, radius = s2_earth_radius_meters())

s2_cell_area_approx(x, radius = s2_earth_radius_meters())

s2_cell_parent(x, level = -1L)

s2_cell_child(x, k)

s2_cell_edge_neighbour(x, k)

s2_cell_contains(x, y)

s2_cell_distance(x, y, radius = s2_earth_radius_meters())

s2_cell_max_distance(x, y, radius = s2_earth_radius_meters())

s2_cell_may_intersect(x, y)
```

```
s2_cell_common_ancestor_level(x, y)

s2_cell_common_ancestor_level_agg(x, na.rm = FALSE)
```

Arguments

x, y	An s2_cell() vector
k	An integer between 0 and 3
radius	The radius to use (e.g., s2_earth_radius_meters())
level	An integer between 0 and 30, inclusive.
na.rm	Remove NAs prior to computing aggregate?

---

s2_cell_union *Create S2 Cell Union vectors*

Description

Create S2 Cell Union vectors

Usage

```
s2_cell_union(x = list())

## S3 method for class 's2_cell_union'
as_s2_geography(x, ...)

as_s2_cell_union(x, ...)

## S3 method for class 's2_cell_union'
as_s2_cell_union(x, ...)

## S3 method for class 's2_cell'
as_s2_cell_union(x, ...)

## S3 method for class 'character'
as_s2_cell_union(x, ...)
```

Arguments

x	A list() of s2_cell() vectors.
...	Passed to S3 methods

Value

An object of class "s2_cell_union".

s2_cell_union_normalize *S2 cell union operators*

Description

S2 cell union operators

Usage

```
s2_cell_union_normalize(x)

s2_cell_union_contains(x, y)

s2_cell_union_intersects(x, y)

s2_cell_union_intersection(x, y)

s2_cell_union_union(x, y)

s2_cell_union_difference(x, y)

s2_covering_cell_ids(
  x,
  min_level = 0,
  max_level = 30,
  max_cells = 8,
  buffer = 0,
  interior = FALSE,
  radius = s2_earth_radius_meters()
)

s2_covering_cell_ids_agg(
  x,
  min_level = 0,
  max_level = 30,
  max_cells = 8,
  buffer = 0,
  interior = FALSE,
  radius = s2_earth_radius_meters(),
  na.rm = FALSE
)
```

Arguments

x, y	An s2_geography or s2_cell_union().

min_level, max_level	The minimum and maximum levels to constrain the covering.
max_cells	The maximum number of cells in the covering. Defaults to 8.
buffer	A distance to buffer outside the geography
interior	Use TRUE to force the covering inside the geography.
radius	The radius to use (e.g., s2_earth_radius_meters())
na.rm	Remove NAs prior to computing aggregate?

---

s2_closest_feature *Matrix Functions*

Description

These functions are similar to accessors and predicates, but instead of recycling x and y to a common length and returning a vector of that length, these functions return a vector of length x with each element i containing information about how the entire vector y relates to the feature at x[i].

Usage

```
s2_closest_feature(x, y)

s2_closest_edges(
  x,
  y,
  k,
  min_distance = -1,
  max_distance = Inf,
  radius = s2_earth_radius_meters()
)

s2_farthest_feature(x, y)

s2_distance_matrix(x, y, radius = s2_earth_radius_meters())

s2_max_distance_matrix(x, y, radius = s2_earth_radius_meters())

s2_contains_matrix(x, y, options = s2_options(model = "open"))

s2_within_matrix(x, y, options = s2_options(model = "open"))

s2_covers_matrix(x, y, options = s2_options(model = "closed"))

s2_covered_by_matrix(x, y, options = s2_options(model = "closed"))

s2_intersects_matrix(x, y, options = s2_options())
```

```
s2_disjoint_matrix(x, y, options = s2_options())

s2_equals_matrix(x, y, options = s2_options())

s2_touches_matrix(x, y, options = s2_options())

s2_dwithin_matrix(x, y, distance, radius = s2_earth_radius_meters())

s2_may_intersect_matrix(x, y, max_edges_per_cell = 50, max_feature_cells = 4)
```

Arguments

x, y	Geography vectors, coerced using as_s2_geography(). x is considered the source, where as y is considered the target.
k	The number of closest edges to consider when searching. Note that in S2 a point is also considered an edge.
min_distance	The minimum distance to consider when searching for edges. This filter is applied after the search is complete (i.e., may cause fewer than k values to be returned).
max_distance	The maximum distance to consider when searching for edges. This filter is applied before the search.
radius	Radius of the earth. Defaults to the average radius of the earth in meters as defined by s2_earth_radius_meters().
options	An s2_options() object describing the polygon/polyline model to use and the snap level.
distance	A distance on the surface of the earth in the same units as radius.
max_edges_per_cell	For s2_may_intersect_matrix(), this values controls the nature of the index on y, with higher values leading to coarser index. Values should be between 10 and 50; the default of 50 is adequate for most use cases, but for specialized operations users may wish to use a lower value to increase performance.
max_feature_cells	For s2_may_intersect_matrix(), this value controls the approximation of x used to identify potential intersections on y. The default value of 4 gives the best performance for most operations, but for specialized operations users may wish to use a higher value to increase performance.

Value

A vector of length x.

See Also

See pairwise predicate functions (e.g., s2_intersects()).

Examples

```
city_names <- c("Vatican City", "San Marino", "Luxembourg")
cities <- s2_data_cities(city_names)
country_names <- s2_data_tbl_countries$name
countries <- s2_data_countries()

# closest feature returns y indices of the closest feature
# for each feature in x
country_names[s2_closest_feature(cities, countries)]

# farthest feature returns y indices of the farthest feature
# for each feature in x
country_names[s2_farthest_feature(cities, countries)]

# use s2_closest_edges() to find the k-nearest neighbours
nearest <- s2_closest_edges(cities, cities, k = 2, min_distance = 0)
city_names
city_names[unlist(nearest)]

# predicate matrices
country_names[s2_intersects_matrix(cities, countries)[[1]]]

# distance matrices
s2_distance_matrix(cities, cities)
s2_max_distance_matrix(cities, countries[1:4])
```

---

s2_contains *S2 Geography Predicates*

Description

These functions operate two geography vectors (pairwise), and return a logical vector.

Usage

```
s2_contains(x, y, options = s2_options(model = "open"))

s2_within(x, y, options = s2_options(model = "open"))

s2_covered_by(x, y, options = s2_options(model = "closed"))

s2_covers(x, y, options = s2_options(model = "closed"))

s2_disjoint(x, y, options = s2_options())

s2_intersects(x, y, options = s2_options())
```

```
s2_equals(x, y, options = s2_options())

s2_intersects_box(
  x,
  lng1,
  lat1,
  lng2,
  lat2,
  detail = 1000,
  options = s2_options()
)

s2_touches(x, y, options = s2_options())

s2_dwithin(x, y, distance, radius = s2_earth_radius_meters())

s2_prepared_dwithin(x, y, distance, radius = s2_earth_radius_meters())
```

Arguments

x, y	geography vectors. These inputs are passed to `as_s2_geography()`, so you can pass other objects (e.g., character vectors of well-known text) directly.
options	An `s2_options()` object describing the polygon/polyline model to use and the snap level.
lng1, lat1, lng2, lat2	A latitude/longitude range
detail	The number of points with which to approximate non-geodesic edges.
distance	A distance on the surface of the earth in the same units as `radius`.
radius	Radius of the earth. Defaults to the average radius of the earth in meters as defined by `s2_earth_radius_meters()`.

Model

The geometry model indicates whether or not a geometry includes its boundaries. Boundaries of line geometries are its end points. OPEN geometries do not contain their boundary (`model = "open"`); CLOSED geometries (`model = "closed"`) contain their boundary; SEMI-OPEN geometries (`model = "semi-open"`) contain half of their boundaries, such that when two polygons do not overlap or two lines do not cross, no point exist that belong to more than one of the geometries. (This latter form, half-closed, is not present in the OpenGIS "simple feature access" (SFA) standard nor DE9-IM on which that is based). The default values for `s2_contains()` (open) and covers/covered_by (closed) correspond to the SFA standard specification of these operators.

See Also

Matrix versions of these predicates (e.g., `s2_intersects_matrix()`).

BigQuery's geography function reference:

- ST_CONTAINS

- ST_COVEREDBY
- ST_COVERS
- ST_DISJOINT
- ST_EQUALS
- ST_INTERSECTS
- ST_INTERSECTSBOX
- ST_TOUCHES
- ST_WITHIN
- ST_DWITHIN

Examples

```
s2_contains(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c("POINT (5 5)", "POINT (-1 1)")
)

s2_within(
  c("POINT (5 5)", "POINT (-1 1)"),
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))"
)

s2_covered_by(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c("POINT (5 5)", "POINT (-1 1)")
)

s2_covers(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c("POINT (5 5)", "POINT (-1 1)")
)

s2_disjoint(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c("POINT (5 5)", "POINT (-1 1)")
)

s2_intersects(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c("POINT (5 5)", "POINT (-1 1)")
)

s2_equals(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c(
    "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
    "POLYGON ((10 0, 10 10, 0 10, 0 0, 10 0))",
    "POLYGON ((-1 -1, 10 0, 10 10, 0 10, -1 -1))"
  )
```

```
)

s2_intersects(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c("POINT (5 5)", "POINT (-1 1)")
)

s2_intersects_box(
  c("POINT (5 5)", "POINT (-1 1)"),
  0, 0, 10, 10
)

s2_touches(
  "POLYGON ((0 0, 0 1, 1 1, 0 0))",
  c("POINT (0 0)", "POINT (0.5 0.75)", "POINT (0 0.5)")
)

s2_dwithin(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c("POINT (5 5)", "POINT (-1 1)"),
  0 # distance in meters
)

s2_dwithin(
  "POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))",
  c("POINT (5 5)", "POINT (-1 1)"),
  1e6 # distance in meters
)
```

`s2_data_example_wkt` *Example Geometries*

Description

These geometries are toy examples useful for testing various coordinate shuffling operations in the s2 package.

Usage

```
s2_data_example_wkt
```

Format

An object of class `list` of length 29.

s2_data_tbl_countries *Low-resolution world boundaries, timezones, and cities*

Description

Well-known binary versions of the Natural Earth low-resolution world boundaries and timezone boundaries.

Usage

```
s2_data_tbl_countries

s2_data_tbl_timezones

s2_data_tbl_cities

s2_data_countries(name = NULL)

s2_data_timezones(utc_offset_min = NULL, utc_offset_max = utc_offset_min)

s2_data_cities(name = NULL)
```

Arguments

name	The name of a country, continent, city, or NULL for all features.
utc_offset_min, utc_offset_max	Minimum and/or maximum timezone offsets.

Format

A data.frame with columns name (character), and geometry (wk_wkb)

An object of class data.frame with 120 rows and 2 columns.

An object of class data.frame with 243 rows and 3 columns.

Source

Natural Earth Data

Examples

```
head(s2_data_countries())
s2_data_countries("Germany")
s2_data_countries("Europe")

head(s2_data_timezones())
s2_data_timezones(-4)
```

```
head(s2_data_cities())
s2_data_cities("Cairo")
```

`s2_earth_radius_meters` *Earth Constants*

Description

According to Yoder (1995), the radius of the earth is 6371.01 km. These functions are used to set the default radis for functions that return a distance or accept a distance as input (e.g., `s2_distance()` and `s2_dwithin()`).

Usage

```
s2_earth_radius_meters()
```

References

Yoder, C.F. 1995. "Astrometric and Geodetic Properties of Earth and the Solar System" in Global Earth Physics, A Handbook of Physical Constants, AGU Reference Shelf 1, American Geophysical Union, Table 2. doi:10.1029/RF001p0001

Examples

```
s2_earth_radius_meters()
```

`s2_geog_point` *Create and Format Geography Vectors*

Description

These functions create and export geography vectors. Unlike the BigQuery geography constructors, these functions do not sanitize invalid or redundant input using `s2_union()`. Note that when creating polygons using `s2_make_polygon()`, rings can be open or closed.

Usage

```
s2_geog_point(longitude, latitude)

s2_make_line(longitude, latitude, feature_id = 1L)

s2_make_polygon(
  longitude,
  latitude,
  feature_id = 1L,
  ring_id = 1L,
  oriented = FALSE,
  check = TRUE
)

s2_geog_from_text(
  wkt_string,
  oriented = FALSE,
  check = TRUE,
  planar = FALSE,
  tessellate_tol_m = s2_tessellate_tol_default()
)

s2_geog_from_wkb(
  wkb_bytes,
  oriented = FALSE,
  check = TRUE,
  planar = FALSE,
  tessellate_tol_m = s2_tessellate_tol_default()
)

s2_as_text(
  x,
  precision = 16,
  trim = TRUE,
  planar = FALSE,
  tessellate_tol_m = s2_tessellate_tol_default()
)

s2_as_binary(
  x,
  endian = wk::wk_platform_endian(),
  planar = FALSE,
  tessellate_tol_m = s2_tessellate_tol_default()
)

s2_tessellate_tol_default()
```

Arguments

longitude, latitude	Vectors of latitude and longitude
feature_id, ring_id	Vectors for which a change in sequential values indicates a new feature or ring. Use factor() to convert from a character vector.
oriented	TRUE if polygon ring directions are known to be correct (i.e., exterior rings are defined counter clockwise and interior rings are defined clockwise).
check	Use check = FALSE to skip error on invalid geometries
wkt_string	Well-known text
planar	Use TRUE to force planar edges in import or export.
tessellate_tol_m	The maximum number of meters to that a point must be moved to satisfy the planar edge constraint.
wkb_bytes	A list() of raw()
x	An object that can be converted to an s2_geography vector
precision	The number of significant digits to export when writing well-known text. If trim = FALSE, the number of digits after the decimal place.
trim	Should trailing zeroes be included after the decimal place?
endian	The endian-ness of the well-known binary. See wk::wkb_translate_wkb().

See Also

See as_s2_geography() for other ways to construct geography vectors.

BigQuery's geography function reference:

- ST_GEOGPOINT
- ST_MAKELINE
- ST_MAKEPOLYGON
- ST_GEOGFROMTEXT
- ST_GEOGFROMWKB
- ST_ASTEXT
- ST_ASBINARY

Examples

```
# create point geographies using coordinate values:
s2_geog_point(-64, 45)

# create line geographies using coordinate values:
s2_make_line(c(-64, 8), c(45, 71))

# optionally, separate features using feature_id:
s2_make_line(
```

```
  c(-64, 8, -27, -27), c(45, 71, 0, 45),
  feature_id = c(1, 1, 2, 2)
)

# create polygon geographies using coordinate values:
# (rings can be open or closed)
s2_make_polygon(c(-45, 8, 0), c(64, 71, 90))

# optionally, separate rings and/or features using
# ring_id and/or feature_id
s2_make_polygon(
  c(20, 10, 10, 30, 45, 30, 20, 20, 40, 20, 45),
  c(35, 30, 10, 5, 20, 20, 15, 25, 40, 45, 30),
  feature_id = c(rep(1, 8), rep(2, 3)),
  ring_id = c(1, 1, 1, 1, 1, 2, 2, 2, 1, 1, 1)
)

# import and export well-known text
(geog <- s2_geog_from_text("POINT (-64 45)"))
s2_as_text(geog)

# import and export well-known binary
(geog <- s2_geog_from_wkb(wk::as_wkb("POINT (-64 45)")))
s2_as_binary(geog)

# import geometry from planar space
s2_geog_from_text(
   "POLYGON ((0 0, 1 0, 0 1, 0 0))",
   planar = TRUE,
   tessellate_tol_m = 1
)

# export geographies into planar space
geog <- s2_make_polygon(c(179, -179, 179), c(10, 10, 11))
s2_as_text(geog, planar = TRUE)

# polygons containing a pole need an extra step
geog <- s2_data_countries("Antarctica")
geom <- s2_as_text(
  s2_intersection(geog, s2_world_plate_carree()),
  planar = TRUE
)
```

`s2_is_collection` *S2 Geography Accessors*

Description

Accessors extract information about geography vectors.

Usage

```
s2_is_collection(x)

s2_is_valid(x)

s2_is_valid_detail(x)

s2_dimension(x)

s2_num_points(x)

s2_is_empty(x)

s2_area(x, radius = s2_earth_radius_meters())

s2_length(x, radius = s2_earth_radius_meters())

s2_perimeter(x, radius = s2_earth_radius_meters())

s2_x(x)

s2_y(x)

s2_distance(x, y, radius = s2_earth_radius_meters())

s2_max_distance(x, y, radius = s2_earth_radius_meters())
```

Arguments

x, y	geography vectors. These inputs are passed to `as_s2_geography()`, so you can pass other objects (e.g., character vectors of well-known text) directly.
radius	Radius of the earth. Defaults to the average radius of the earth in meters as defined by `s2_earth_radius_meters()`.

See Also

BigQuery's geography function reference:

- ST_ISCOLLECTION
- ST_DIMENSION
- ST_NUMPOINTS
- ST_ISEMPTY
- ST_AREA
- ST_LENGTH
- ST_PERIMETER
- ST_X

- ST_Y
- ST_DISTANCE
- ST_MAXDISTANCE

Examples

```
# s2_is_collection() tests for multiple geometries in one feature
s2_is_collection(c("POINT (-64 45)", "MULTIPOINT ((-64 45), (8 72))"))

# s2_dimension() returns 0 for point, 1  for line, 2 for polygon
s2_dimension(
  c(
    "GEOMETRYCOLLECTION EMPTY",
    "POINT (-64 45)",
    "LINESTRING (-64 45, 8 72)",
    "POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))",
    "GEOMETRYCOLLECTION (POINT (-64 45), LINESTRING (-64 45, 8 72))"
   )
)

# s2_num_points() counts points
s2_num_points(c("POINT (-64 45)", "LINESTRING (-64 45, 8 72)"))

# s2_is_empty tests for emptiness
s2_is_empty(c("POINT (-64 45)", "POINT EMPTY"))

# calculate area, length, and perimeter
s2_area("POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))")
s2_perimeter("POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))")
s2_length(s2_boundary("POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))"))

# extract x and y coordinates from points
s2_x(c("POINT (-64 45)", "POINT EMPTY"))
s2_y(c("POINT (-64 45)", "POINT EMPTY"))

# calculate minimum and maximum distance between two geometries
s2_distance(
  "POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))",
  "POINT (-64 45)"
)
s2_max_distance(
  "POLYGON ((0 0, 0 10, 10 10, 10 0, 0 0))",
  "POINT (-64 45)"
)
```

---

s2_lnglat *Create an S2 LngLat Vector*

Description

This class represents a latitude and longitude on the Earth's surface. Most calculations in S2 convert this to a `as_s2_point()`, which is a unit vector representation of this value.

Usage

```
s2_lnglat(lng, lat)

as_s2_lnglat(x, ...)

## Default S3 method:
as_s2_lnglat(x, ...)

## S3 method for class 'wk_xy'
as_s2_lnglat(x, ...)

## S3 method for class 'wk_xyz'
as_s2_lnglat(x, ...)
```

Arguments

`lat, lng`	Vectors of latitude and longitude values in degrees.
`x`	A `s2_lnglat()` vector or an object that can be coerced to one.
`...`	Unused

Value

An object with class s2_lnglat

Examples

```
s2_lnglat(45, -64) # Halifax, Nova Scotia!
as.data.frame(s2_lnglat(45, -64))
```

`s2_options` *Geography Operation Options*

Description

These functions specify defaults for options used to perform operations and construct geometries. These are used in predicates (e.g., `s2_intersects()`), and boolean operations (e.g., `s2_intersection()`) to specify the model for containment and how new geometries should be constructed.

Usage

```
s2_options(
  model = NULL,
  snap = s2_snap_identity(),
  snap_radius = -1,
  duplicate_edges = FALSE,
  edge_type = "directed",
  validate = FALSE,
  polyline_type = "path",
  polyline_sibling_pairs = "keep",
  simplify_edge_chains = FALSE,
  split_crossing_edges = FALSE,
  idempotent = FALSE,
  dimensions = c("point", "polyline", "polygon")
)

s2_snap_identity()

s2_snap_level(level)

s2_snap_precision(precision)

s2_snap_distance(distance)
```

Arguments

model	One of 'open', 'semi-open' (default for polygons), or 'closed' (default for polylines). See section 'Model'
snap	Use `s2_snap_identity()`, `s2_snap_distance()`, `s2_snap_level()`, or `s2_snap_precision()` to specify how or if coordinate rounding should occur.
snap_radius	As opposed to the snap function, which specifies the maximum distance a vertex should move, the snap radius (in radians) sets the minimum distance between vertices of the output that don't cause vertices to move more than the distance specified by the snap function. This can be used to simplify the result of a boolean operation. Use -1 to specify that any minimum distance is acceptable.
duplicate_edges	Use `TRUE` to keep duplicate edges (e.g., duplicate points).
edge_type	One of 'directed' (default) or 'undirected'.
validate	Use `TRUE` to validate the result from the builder.
polyline_type	One of 'path' (default) or 'walk'. If 'walk', polylines that backtrack are preserved.
polyline_sibling_pairs	One of 'discard' (default) or 'keep'.
simplify_edge_chains	Use `TRUE` to remove vertices that are within `snap_radius` of the original vertex.
split_crossing_edges	Use `TRUE` to split crossing polyline edges when creating geometries.

idempotent	Use FALSE to apply snap even if snapping is not necessary to satisfy vertex constraints.
dimensions	A combination of 'point', 'polyline', and/or 'polygon' that can used to constrain the output of s2_rebuild() or a boolean operation.
level	A value from 0 to 30 corresponding to the cell level at which snapping should occur.
precision	A number by which coordinates should be multiplied before being rounded. Rounded to the nearest exponent of 10.
distance	A distance (in radians) denoting the maximum distance a vertex should move in the snapping process.

Model

The geometry model indicates whether or not a geometry includes its boundaries. Boundaries of line geometries are its end points. OPEN geometries do not contain their boundary (model = "open"); CLOSED geometries (model = "closed") contain their boundary; SEMI-OPEN geometries (model = "semi-open") contain half of their boundaries, such that when two polygons do not overlap or two lines do not cross, no point exist that belong to more than one of the geometries. (This latter form, half-closed, is not present in the OpenGIS "simple feature access" (SFA) standard nor DE9-IM on which that is based). The default values for s2_contains() (open) and covers/covered_by (closed) correspond to the SFA standard specification of these operators.

Examples

```
# use s2_options() to specify containment models, snap level
# layer creation options, and builder options
s2_options(model = "closed", snap = s2_snap_level(30))
```

---

s2_plot *Plot S2 Geographies*

Description

Plot S2 Geographies

Usage

```
s2_plot(
  x,
  ...,
  asp = 1,
  xlab = "",
  ylab = "",
  rule = "evenodd",
  add = FALSE,
```

```
  plot_hemisphere = FALSE,
  simplify = TRUE,
  centre = NULL
)
```

Arguments

x	A wkb() or wkt()
...	Passed to plotting functions for features: graphics::points() for point and multipoint geometries, graphics::lines() for linestring and multilinestring geometries, and graphics::polypath() for polygon and multipolygon geometries.
asp, xlab, ylab	Passed to graphics::plot()
rule	The rule to use for filling polygons (see graphics::polypath())
add	Should a new plot be created, or should handleable be added to the existing plot?
plot_hemisphere	Plot the outline of the earth
simplify	Use FALSE to skip the simplification step
centre	The longitude/latitude point of the centre of the orthographic projection

Value

The input, invisibly

Examples

```
s2_plot(s2_data_countries())
s2_plot(s2_data_cities(), add = TRUE)
```

---

s2_point *Create an S2 Point Vector*

Description

In S2 terminology, a "point" is a 3-dimensional unit vector representation of an s2_point(). Internally, all s2 objects are stored as 3-dimensional unit vectors.

Usage

```
s2_point(x, y, z)

s2_point_crs()

as_s2_point(x, ...)

## Default S3 method:
as_s2_point(x, ...)

## S3 method for class 'wk_xy'
as_s2_point(x, ...)

## S3 method for class 'wk_xyz'
as_s2_point(x, ...)
```

Arguments

x, y, z	Vectors of latitude and longitude values in degrees.
...	Unused

Value

An object with class s2_point

Examples

```
point <- s2_lnglat(-64, 45) # Halifax, Nova Scotia!
as_s2_point(point)
as.data.frame(as_s2_point(point))
```

---

s2_project *Linear referencing*

Description

Linear referencing

Usage

```
s2_project(x, y, radius = s2_earth_radius_meters())

s2_project_normalized(x, y)

s2_interpolate(x, distance, radius = s2_earth_radius_meters())

s2_interpolate_normalized(x, distance_normalized)
```

Arguments

x	A simple polyline geography vector
y	A simple point geography vector. The point will be snapped to the nearest point on x for the purposes of interpolation.
radius	Radius of the earth. Defaults to the average radius of the earth in meters as defined by s2_earth_radius_meters().
distance	A distance along x in radius units.
distance_normalized	A distance normalized to s2_length() of x.

Value

- s2_interpolate() returns the point on x, distance meters along the line.
- s2_interpolate_normalized() returns the point on x interpolated to a fraction along the line.
- s2_project() returns the distance that point occurs along x.
- s2_project_normalized() returns the distance_normalized along x where point occurs.

Examples

```
s2_project_normalized("LINESTRING (0 0, 0 90)", "POINT (0 22.5)")
s2_project("LINESTRING (0 0, 0 90)", "POINT (0 22.5)")
s2_interpolate_normalized("LINESTRING (0 0, 0 90)", 0.25)
s2_interpolate("LINESTRING (0 0, 0 90)", 2501890)
```

---

wk_handle.s2_geography

Low-level wk filters and handlers

Description

Low-level wk filters and handlers

Usage

```
## S3 method for class 's2_geography'
wk_handle(
  handleable,
  handler,
  ...,
  s2_projection = s2_projection_plate_carree(),
  s2_tessellate_tol = Inf
)
```

```
s2_geography_writer(
  oriented = FALSE,
  check = TRUE,
  projection = s2_projection_plate_carree(),
  tessellate_tol = Inf
)

## S3 method for class 's2_geography'
wk_writer(handleable, ...)

s2_trans_point()

s2_trans_lnglat()

s2_projection_plate_carree(x_scale = 180)

s2_projection_mercator(x_scale = 20037508.3427892)

s2_hemisphere(centre)

s2_world_plate_carree(epsilon_east_west = 0, epsilon_north_south = 0)

s2_projection_orthographic(centre = s2_lnglat(0, 0))
```

Arguments

handleable	A geometry vector (e.g., wkb(), wkt(), xy(), rct(), or sf::st_sfc()) for which wk_handle() is defined.
handler	A wk_handler object.
...	Passed to the wk_handle() method.
oriented	TRUE if polygon ring directions are known to be correct (i.e., exterior rings are defined counter clockwise and interior rings are defined clockwise).
check	Use check = FALSE to skip error on invalid geometries
projection, s2_projection	One of s2_projection_plate_carree() or s2_projection_mercator()
tessellate_tol, s2_tessellate_tol	An angle in radians. Points will not be added if a line segment is within this distance of a point.
x_scale	The maximum x value of the projection
centre	The center point of the orthographic projection
epsilon_east_west, epsilon_north_south	Use a positive number to define the edges of a Cartesian world slightly inward from -180, -90, 180, 90. This may be used to define a world outline for a projection where projecting at the extreme edges of the earth results in a non-finite value.

Value

- `s2_projection_plate_carree()`, `s2_projection_mercator()`: An external pointer to an S2 projection.

Index

∗ **datasets**
    s2_data_example_wkt, 18
    s2_data_tbl_countries, 19

as_s2_cell *(*s2_cell*)*, 8
as_s2_cell_union *(*s2_cell_union*)*, 11
as_s2_geography, 2
as_s2_geography(), *3*, *5*, *14*, *16*, *22*, *24*
as_s2_geography.s2_cell_union *(*s2_cell_union*)*, 11
as_s2_lnglat *(*s2_lnglat*)*, 25
as_s2_point *(*s2_point*)*, 29
as_s2_point(), *26*
as_wkb.s2_geography *(*as_s2_geography*)*, 2
as_wkt.s2_geography *(*as_s2_geography*)*, 2

double(), *9*

factor(), *22*

geography vectors, *5*, *16*, *20*, *23*, *24*
graphics::lines(), *29*
graphics::plot(), *29*
graphics::points(), *29*
graphics::polypath(), *29*

new_s2_cell *(*s2_cell*)*, 8

rct(), *32*

s2_area *(*s2_is_collection*)*, 23
s2_as_binary *(*s2_geog_point*)*, 20
s2_as_binary(), *3*
s2_as_text *(*s2_geog_point*)*, 20
s2_as_text(), *3*
s2_boundary, 4
s2_bounds_cap, 7
s2_bounds_cap(), *7*, *8*
s2_bounds_rect *(*s2_bounds_cap*)*, 7
s2_bounds_rect(), *7*, *8*
s2_buffer_cells *(*s2_boundary*)*, 4
s2_cell, 8
s2_cell(), *11*
s2_cell_area *(*s2_cell_is_valid*)*, 10
s2_cell_area_approx *(*s2_cell_is_valid*)*, 10
s2_cell_boundary *(*s2_cell_is_valid*)*, 10
s2_cell_center *(*s2_cell_is_valid*)*, 10
s2_cell_child *(*s2_cell_is_valid*)*, 10
s2_cell_common_ancestor_level *(*s2_cell_is_valid*)*, 10
s2_cell_common_ancestor_level_agg *(*s2_cell_is_valid*)*, 10
s2_cell_contains *(*s2_cell_is_valid*)*, 10
s2_cell_debug_string *(*s2_cell_is_valid*)*, 10
s2_cell_distance *(*s2_cell_is_valid*)*, 10
s2_cell_edge_neighbour *(*s2_cell_is_valid*)*, 10
s2_cell_invalid *(*s2_cell*)*, 8
s2_cell_is_face *(*s2_cell_is_valid*)*, 10
s2_cell_is_leaf *(*s2_cell_is_valid*)*, 10
s2_cell_is_valid, 10
s2_cell_level *(*s2_cell_is_valid*)*, 10
s2_cell_max_distance *(*s2_cell_is_valid*)*, 10
s2_cell_may_intersect *(*s2_cell_is_valid*)*, 10
s2_cell_parent *(*s2_cell_is_valid*)*, 10
s2_cell_polygon *(*s2_cell_is_valid*)*, 10
s2_cell_sentinel *(*s2_cell*)*, 8
s2_cell_to_lnglat *(*s2_cell_is_valid*)*, 10
s2_cell_union, 11
s2_cell_union(), *12*
s2_cell_union_contains *(*s2_cell_union_normalize*)*, 12
s2_cell_union_difference *(*s2_cell_union_normalize*)*, 12
s2_cell_union_intersection *(*s2_cell_union_normalize*)*, 12

s2_cell_union_intersects
(s2_cell_union_normalize), 12
s2_cell_union_normalize, 12
s2_cell_union_union
(s2_cell_union_normalize), 12
s2_cell_vertex (s2_cell_is_valid), 10
s2_centroid (s2_boundary), 4
s2_centroid_agg (s2_boundary), 4
s2_closest_edges (s2_closest_feature),
13
s2_closest_feature, 13
s2_closest_point (s2_boundary), 4
s2_contains, 15
s2_contains(), 5, 16, 28
s2_contains_matrix
(s2_closest_feature), 13
s2_convex_hull (s2_boundary), 4
s2_convex_hull_agg (s2_boundary), 4
s2_coverage_union_agg (s2_boundary), 4
s2_covered_by (s2_contains), 15
s2_covered_by_matrix
(s2_closest_feature), 13
s2_covering_cell_ids
(s2_cell_union_normalize), 12
s2_covering_cell_ids_agg
(s2_cell_union_normalize), 12
s2_covers (s2_contains), 15
s2_covers_matrix (s2_closest_feature),
13
s2_data_cities (s2_data_tbl_countries),
19
s2_data_countries
(s2_data_tbl_countries), 19
s2_data_example_wkt, 18
s2_data_tbl_cities
(s2_data_tbl_countries), 19
s2_data_tbl_countries, 19
s2_data_tbl_timezones
(s2_data_tbl_countries), 19
s2_data_timezones
(s2_data_tbl_countries), 19
s2_difference (s2_boundary), 4
s2_dimension (s2_is_collection), 23
s2_disjoint (s2_contains), 15
s2_disjoint_matrix
(s2_closest_feature), 13
s2_distance (s2_is_collection), 23
s2_distance(), 20
s2_distance_matrix
(s2_closest_feature), 13
s2_dwithin (s2_contains), 15
s2_dwithin(), 20
s2_dwithin_matrix (s2_closest_feature),
13
s2_earth_radius_meters, 20
s2_earth_radius_meters(), 5, 11, 13, 14,
16, 24, 31
s2_equals (s2_contains), 15
s2_equals_matrix (s2_closest_feature),
13
s2_farthest_feature
(s2_closest_feature), 13
s2_geog_from_text (s2_geog_point), 20
s2_geog_from_text(), 3
s2_geog_from_wkb (s2_geog_point), 20
s2_geog_from_wkb(), 3
s2_geog_point, 20
s2_geog_point(), 3
s2_geography, 12
s2_geography (as_s2_geography), 2
s2_geography(), 8
s2_geography_writer
(wk_handle.s2_geography), 31
s2_hemisphere (wk_handle.s2_geography),
31
s2_interpolate (s2_project), 30
s2_interpolate_normalized (s2_project),
30
s2_intersection (s2_boundary), 4
s2_intersection(), 26
s2_intersects (s2_contains), 15
s2_intersects(), 14, 26
s2_intersects_box (s2_contains), 15
s2_intersects_matrix
(s2_closest_feature), 13
s2_intersects_matrix(), 16
s2_is_collection, 23
s2_is_empty (s2_is_collection), 23
s2_is_valid (s2_is_collection), 23
s2_is_valid_detail (s2_is_collection),
23
s2_length (s2_is_collection), 23
s2_length(), 31
s2_lnglat, 25
s2_lnglat(), 26
s2_make_line (s2_geog_point), 20

s2_make_line(), *3*
s2_make_polygon *(*s2_geog_point*)*, 20
s2_make_polygon(), *3*, *20*
s2_max_distance *(*s2_is_collection*)*, 23
s2_max_distance_matrix *(*s2_closest_feature*)*, 13
s2_may_intersect_matrix *(*s2_closest_feature*)*, 13
s2_may_intersect_matrix(), *14*
s2_minimum_clearance_line_between *(*s2_boundary*)*, 4
s2_num_points *(*s2_is_collection*)*, 23
s2_options, 26
s2_options(), *5*, *14*, *16*
s2_perimeter *(*s2_is_collection*)*, 23
s2_plot, 28
s2_point, 29
s2_point(), *29*
s2_point_crs *(*s2_point*)*, 29
s2_point_on_surface *(*s2_boundary*)*, 4
s2_prepared_dwithin *(*s2_contains*)*, 15
s2_project, 30
s2_project_normalized *(*s2_project*)*, 30
s2_projection_mercator *(*wk_handle.s2_geography*)*, 31
s2_projection_mercator(), *32*
s2_projection_orthographic *(*wk_handle.s2_geography*)*, 31
s2_projection_plate_carree *(*wk_handle.s2_geography*)*, 31
s2_projection_plate_carree(), *32*
s2_rebuild *(*s2_boundary*)*, 4
s2_rebuild(), *28*
s2_rebuild_agg *(*s2_boundary*)*, 4
s2_simplify *(*s2_boundary*)*, 4
s2_snap_distance *(*s2_options*)*, 26
s2_snap_identity *(*s2_options*)*, 26
s2_snap_level *(*s2_options*)*, 26
s2_snap_precision *(*s2_options*)*, 26
s2_snap_to_grid *(*s2_boundary*)*, 4
s2_sym_difference *(*s2_boundary*)*, 4
s2_tessellate_tol_default *(*s2_geog_point*)*, 20
s2_touches *(*s2_contains*)*, 15
s2_touches_matrix *(*s2_closest_feature*)*, 13
s2_trans_lnglat *(*wk_handle.s2_geography*)*, 31
s2_trans_point *(*wk_handle.s2_geography*)*, 31
s2_union *(*s2_boundary*)*, 4
s2_union(), *20*
s2_union_agg *(*s2_boundary*)*, 4
s2_within *(*s2_contains*)*, 15
s2_within_matrix *(*s2_closest_feature*)*, 13
s2_world_plate_carree *(*wk_handle.s2_geography*)*, 31
s2_x *(*s2_is_collection*)*, 23
s2_y *(*s2_is_collection*)*, 23
sf::st_sfc(), *32*

vctrs, *3*

wk::wkb_translate_wkb(), *22*
wk_handle(), *32*
wk_handle.s2_geography, 31
wk_handler, *32*
wk_writer.s2_geography *(*wk_handle.s2_geography*)*, 31
wkb(), *29*, *32*
wkt(), *29*, *32*

xy(), *32*