Package: sx (via r-universe)

May 25, 2026

Title Scalable Spatial Data Analysis Using 'SedonaDB'

Version 0.0.0.9000

Description Provides scalable spatial operations on vector and raster data using 'Apache SedonaDB' ('DataFusion'-based spatial engine) as backend. Enables efficient processing of large spatial datasets without loading all data into 'R' memory, leveraging 'DuckDB' and 'Arrow' for high-performance I/O.

License MIT + file LICENSE

URL https://github.com/e-kotov/sx, https://www.ekotov.pro/sx/

BugReports https://github.com/e-kotov/sx/issues

SystemRequirements PROJ (>= 4.8.0), GDAL (>= 2.0.1), GEOS (>= 3.4.0)

Depends R (>= 4.1.0)

Imports cli, DBI, dbplyr, dplyr, duckdb, geoarrow, glue, jsonlite, lifecycle, nanoarrow, rlang, sedonadb, sf, tibble, tidyselect, withr, wk

Suggests areal, arrow, pkgdown, testthat (>= 3.0.0), uuid, yaml, lwgeom

Additional_repositories https://apache.r-universe.dev

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Repository https://e-kotov.r-universe.dev

Date/Publication 2026-01-25 22:32:14 UTC

RemoteUrl https://github.com/e-kotov/sx

RemoteRef HEAD

RemoteSha cf364750bd392770d4bb2cc1850e0e14b3125a95

Contents

`sx_as_view` *Register object as a named SedonaDB view (Lazy)*

Description

Registers an `sf` object, `sedonadb_dataframe`, or `dbplyr` query as a named **view** in SedonaDB. This operation is **lazy**; it does not move data until a query is executed.

Usage

```
sx_as_view(x, name = NULL, overwrite = TRUE, verbosity = NULL)
```

Arguments

x	A sedonadb_dataframe, sf object, or view name (character) in SedonaDB to register..
name	Character. The name to register the view as. If NULL (default), a unique name is generated.
overwrite	Logical. If TRUE, overwrites any existing view with the same name. Default is FALSE.
verbosity	Character or NULL. Controls message output for this function call. • "quiet": Suppress all informational messages. • "info": Show standard progress and status messages. • "debug": Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.

Value

A sedonadb_dataframe pointing to the registered view (invisibly).

See Also

sx_create_table() to materialize data into memory.

---

sx_buffer *Compute buffer around geometries*

Description

Compute a buffer around this geometry.

Usage

```
sx_buffer(
  x,
  dist,
  output = NULL,
  view_name = NULL,
  verbosity = NULL,
  use_s2 = NULL,
  ...
)
```

Arguments

x	Input object (sf, sedonadb_dataframe, or character view name).
dist	A single numeric value representing the buffer distance. Note: SedonaDB currently does not support column-based distances for buffering.
output	Character or NULL. Output type: sedonadb_dataframe (default), sf, tibble, geoarrow, or raw. If NULL, uses getOption("sx.output_type", "sedonadb_dataframe"). Output types: • sedonadb_dataframe: Lazy data frame (no collection). • sf: Materialized sf object. • tibble: Tibble without geometry. • geoarrow: Tibble with geoarrow_vctr geometry (Arrow-native). • raw: Tibble with geometry as raw WKB bytes (for database import).
view_name	Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view. Not all backends support named views. Check backend-specific documentation for availability.
verbosity	Character or NULL. Controls message output for this function call. • "quiet": Suppress all informational messages. • "info": Show standard progress and status messages. • "debug": Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.
use_s2	Logical or NULL. Controls spherical geometry (S2) for this operation. • TRUE: Use S2 spherical geometry (accurate for geographic coordinates). • FALSE: Use planar geometry (faster, appropriate for projected CRS). • NULL (default): Uses the global sx_use_s2() setting.
...	Ignored. Used to catch and warn about unsupported sf arguments.

Value

Result (type depends on output)

---

sx_centroid *Compute centroid of geometries*

Description

Compute the centroid of this geometry.

Usage

```
sx_centroid(
  x,
  of_largest_polygon = FALSE,
  output = NULL,
  view_name = NULL,
  verbosity = NULL,
  ...
)
```

Arguments

x	Input object (sf, sedonadb_dataframe, or character view name).
of_largest_polygon	Logical. If TRUE, returns the centroid of the largest polygon of a multipolygon. (Not yet supported by SedonaDB backend, ignored with warning if TRUE).
output	Character or NULL. Output type: sedonadb_dataframe (default), sf, tibble, geoarrow, or raw. If NULL, uses getOption("sx.output_type", "sedonadb_dataframe"). Output types: • sedonadb_dataframe: Lazy data frame (no collection). • sf: Materialized sf object. • tibble: Tibble without geometry. • geoarrow: Tibble with geoarrow_vctr geometry (Arrow-native). • raw: Tibble with geometry as raw WKB bytes (for database import).
view_name	Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view. Not all backends support named views. Check backend-specific documentation for availability.
verbosity	Character or NULL. Controls message output for this function call. • "quiet": Suppress all informational messages. • "info": Show standard progress and status messages. • "debug": Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.
...	Ignored. Used to catch and warn about unsupported sf arguments.

Value

Result (type depends on output)

`sx_collect` *Collect SedonaDB result to sf with Safety Guardrails*

Description

Materializes a lazy sedonadb_dataframe into an R sf object. Checks row count against a safety threshold before downloading to prevent crashes from massive datasets.

Usage

```
sx_collect(x, force = FALSE, verbosity = NULL)
```

Arguments

`x`	A `sedonadb_dataframe`, `sf` object, or view name (character) in SedonaDB to be collected.
`force`	Logical. If `TRUE`, bypasses the row count safety check when collecting data via `[sx_collect()]`. Use with caution on large datasets.
`verbosity`	Character or NULL. Controls message output for this function call. • `"quiet"`: Suppress all informational messages. • `"info"`: Show standard progress and status messages. • `"debug"`: Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.

Value

An sf object

Examples

```
library(sf)

# Load sample data
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
box <- st_bbox(nc[1:10, ]) |> st_as_sfc() |> st_as_sf()

# ----------------------------------------------------------------
# Example 1: Collect from sedonadb_dataframe (lazy result)
# ----------------------------------------------------------------
lazy_result <- sx_filter(nc, box, output = "lazy")
sf_result <- sx_collect(lazy_result)
class(sf_result)  # "sf"

# ----------------------------------------------------------------
# Example 2: Collect from a named view
# ----------------------------------------------------------------
```

```
sx_as_view(nc, "nc_data")
sf_from_view <- sx_collect("nc_data")

# ---
# Note: sx_collect has a safety limit (default 1M rows).
# Use force = TRUE to bypass if you're sure about memory capacity:
# sx_collect(large_lazy_result, force = TRUE)
```

`sx_create_table` *Materialize a view into a memory table*

Description

Forces computation of a lazy view or dataframe and stores the result as a materialized **table** in SedonaDB memory (Apache Arrow `MemTable`).

This is useful for caching intermediate results to speed up subsequent queries.

Usage

```
sx_create_table(x, name = NULL, overwrite = TRUE, verbosity = NULL)
```

Arguments

`x`	A `sedonadb_dataframe`, `sf` object, or view name (character) in SedonaDB to materialize.. Can be a view name (character) or `sedonadb_dataframe`.
`name`	Character. The name to register the table as. If `NULL` (default), a unique name is generated.
`overwrite`	Logical. If `TRUE`, overwrites any existing view with the same name. Default is `FALSE`.
`verbosity`	Character or NULL. Controls message output for this function call. • `"quiet"`: Suppress all informational messages. • `"info"`: Show standard progress and status messages. • `"debug"`: Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.

Value

A `sedonadb_dataframe` pointing to the materialized table (invisibly).

Examples

```
## Not run:
# 1. Define a lazy operation
lazy_df <- sx_buffer("input_view", 10)

# 2. Materialize it into memory as 'buffered_table'
sx_create_table(lazy_df, "buffered_table")

## End(Not run)
```

`sx_crs` *Get CRS from a SedonaDB view or sedonadb_dataframe*

Description

Extracts the Coordinate Reference System (CRS) from a SedonaDB view name or `sedonadb_dataframe` by reading the Arrow schema metadata.

This function returns a standard `sf::crs` object, ensuring full compatibility with the `sf` ecosystem.

Usage

```
sx_crs(x, verbosity = NULL)
```

Arguments

`x`	A `sedonadb_dataframe`, `sf` object, or view name (character) in SedonaDB for which to extract the CRS.
`verbosity`	Character or NULL. Controls message output for this function call. • `"quiet"`: Suppress all informational messages. • `"info"`: Show standard progress and status messages. • `"debug"`: Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.

Value

An object of class `crs` (from the `sf` package).

Examples

```
x <- sx_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
sx_crs(x)
```

sx_dplyr *Dplyr verbs for SedonaDB*

Description

sx implements these dplyr::dplyr verbs for sedonadb_dataframe objects. See the documentation of the original generic functions for usage details.

These methods translate R expressions to SQL queries executed by SedonaDB (DataFusion).

Supported Verbs:

- dplyr::select(): Choose columns.
- dplyr::rename(): Rename columns.
- dplyr::mutate(): Add or modify columns.
- dplyr::filter(): Filter rows.
- dplyr::arrange(): Sort rows.
- dplyr::distinct(): Keep unique rows (.keep_all = TRUE is not supported).
- dplyr::pull(): Extract a single column.
- dplyr::collect(): Force computation and return R object.

sx_dplyr_joins *Dplyr joins for SedonaDB*

Description

sx implements dplyr::dplyr joins for sedonadb_dataframe objects. See dplyr::left_join() and dplyr::inner_join() for details.

Supported Joins:

- dplyr::left_join()
- dplyr::inner_join()

`sx_drivers` *Get list of supported spatial drivers and file formats*

Description

Returns a data frame of spatial drivers supported by the current backend. For the DuckDB reader, this includes all GDAL-supported formats.

Usage

```
sx_drivers(verbosity = NULL)
```

Arguments

`verbosity`	Character or NULL. Controls message output for this function call. • `"quiet"`: Suppress all informational messages. • `"info"`: Show standard progress and status messages. • `"debug"`: Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.

Value

A data frame with columns:

- `short_name`: Driver short name (e.g., "GPKG")
- `long_name`: Descriptive name (e.g., "GeoPackage")
- `can_read`: Logical. Whether the driver supports reading.
- `can_write`: Logical. Whether the driver supports writing.
- `can_create`: Logical. Whether the driver supports creating new files.

Examples

```
# List all supported drivers
drivers <- sx_drivers()
head(drivers)

# Check for specific driver
"GPKG" %in% drivers$short_name
```

sx_drop_view *Drop a registered view or table*

Description

Drop a registered view or table

Usage

```
sx_drop_view(name)
```

Arguments

name	Character. Name of the view/table to drop.

Value

Invisibly returns the name.

sx_duckdb_to_sedona *Ingest data from DuckDB to SedonaDB*

Description

Ingests a DuckDB table, view, or dbplyr query into SedonaDB. This operation uses zero-copy Arrow streaming to transfer data efficiently between DuckDB and SedonaDB.

Usage

```
sx_duckdb_to_sedona(
  data,
  conn = NULL,
  name = NULL,
  materialize = TRUE,
  verbosity = NULL
)
```

Arguments

data	A tbl_duckdb_connection (from dbplyr), or a character string representing a table/view name in DuckDB.
conn	A duckdb_connection object. Required if data is a character string.
name	Character string. Optional name for the registered view in SedonaDB. If NULL, a temporary name is generated.

materialize — Logical. If TRUE (default), force materialization of the data in SedonaDB to complete the transfer. If FALSE, returns a lazy pointer (safer for huge datasets, but source connection must remain open).

verbosity — Character or NULL. Controls message output for this function call.

- "quiet": Suppress all informational messages.
- "info": Show standard progress and status messages.
- "debug": Show additional diagnostic messages for troubleshooting.

If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.

Value

A sedonadb_dataframe.

Examples

```
## Not run:
library(sx)
library(dplyr)
library(dbplyr)
library(duckdb)

con <- dbConnect(duckdb())
dbExecute(con, "INSTALL spatial; LOAD spatial;")

# Create a table in DuckDB
dbExecute(con, "CREATE TABLE points (id INTEGER, geom GEOMETRY)")
dbExecute(con, "INSERT INTO points VALUES (1, ST_Point(0,0))")

# Option 1: Ingest from dbplyr object (connection auto-detected)
tbl_points <- tbl(con, "points")
sdf <- sx_duckdb_to_sedona(tbl_points)

# Option 2: Ingest from table name (requires connection)
sdf_2 <- sx_duckdb_to_sedona("points", conn = con)

# Option 3: Ingest and register as named view
sx_duckdb_to_sedona("points", conn = con, name = "sedona_points_view")

dbDisconnect(con)

## End(Not run)
```

---

sx_envelope — *Compute envelope (bounding box) of geometries*

Description

Compute the envelope (bounding box) of this geometry.

Usage

```
sx_envelope(x, output = NULL, view_name = NULL, verbosity = NULL, ...)
```

Arguments

x	Input object (sf, sedonadb_dataframe, or character view name).
output	Character or NULL. Output type: `sedonadb_dataframe` (default), `sf`, `tibble`, `geoarrow`, or `raw`. If NULL, uses `getOption("sx.output_type", "sedonadb_dataframe")`. Output types: • `sedonadb_dataframe`: Lazy data frame (no collection). • `sf`: Materialized sf object. • `tibble`: Tibble without geometry. • `geoarrow`: Tibble with `geoarrow_vctr` geometry (Arrow-native). • `raw`: Tibble with geometry as raw WKB bytes (for database import).
view_name	Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view. Not all backends support named views. Check backend-specific documentation for availability.
verbosity	Character or NULL. Controls message output for this function call. • `"quiet"`: Suppress all informational messages. • `"info"`: Show standard progress and status messages. • `"debug"`: Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.
...	Ignored. Used to catch and warn about unsupported sf arguments.

Value

Result (type depends on `output`)

`sx_filter` *Performs spatial filter of two geometries using SedonaDB*

Description

Filters an `sf` object spatially based on its relationship with another `sf` object or geometry. All calculations are performed in SedonaDB for efficiency. Supports lazy evaluation returning `sedonadb_dataframe` objects.

Usage

```
sx_filter(
  x,
  y,
  predicate = "intersects",
  distance = NULL,
  output = NULL,
  view_name = NULL,
  verbosity = NULL,
  use_s2 = NULL,
  ...
)
```

Arguments

x	A sedonadb_dataframe, sf object, or view name (character) in SedonaDB representing features to filter.
y	A sedonadb_dataframe, sf object, or view name (character) in SedonaDB representing the filtering geometry.
predicate	Character. Spatial predicate to use. One of: intersects (default), contains, within, touches, crosses, overlaps, covers, covered_by, disjoint, equals, contains_properly, or dwithin.
distance	Numeric (optional). Distance threshold for the dwithin predicate. Required when predicate = "dwithin".
output	Character or NULL. Output type: sedonadb_dataframe (default), sf, tibble, geoarrow, or raw. If NULL, uses getOption("sx.output_type", "sedonadb_dataframe"). Output types: • sedonadb_dataframe: Lazy data frame (no collection). • sf: Materialized sf object. • tibble: Tibble without geometry. • geoarrow: Tibble with geoarrow_vctr geometry (Arrow-native). • raw: Tibble with geometry as raw WKB bytes (for database import).
view_name	Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view. Not all backends support named views. Check backend-specific documentation for availability.
verbosity	Character or NULL. Controls message output for this function call. • "quiet": Suppress all informational messages. • "info": Show standard progress and status messages. • "debug": Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.
use_s2	Logical or NULL. Controls spherical geometry (S2) for this operation. • TRUE: Use S2 spherical geometry (accurate for geographic coordinates).

- FALSE: Use planar geometry (faster, appropriate for projected CRS).
- NULL (default): Uses the global sx_use_s2() setting.

... Additional arguments passed to methods.

Value

An sf object or sedonadb_dataframe (depending on output).

Examples

```
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
box <- st_bbox(nc[1:10, ]) |> st_as_sfc() |> st_as_sf()

# ------------------------------------------------------------------
# Example 1: Using sf objects (most common)
# ------------------------------------------------------------------
# Default lazy return
lazy_result <- sx_filter(nc, box, predicate = "intersects")

# Materialize to sf
sf_result <- sx_collect(lazy_result)

# Or request sf directly
sf_result <- sx_filter(nc, box, output = "sf")

# ------------------------------------------------------------------
# Example 2: Using sedonadb_dataframe (chain lazy operations)
# ------------------------------------------------------------------
# First filter returns lazy result
step1 <- sx_filter(nc, box, predicate = "intersects", output = "lazy")

# Chain with another operation (filter by a different geometry)
box2 <- st_bbox(nc[20:25, ]) |> st_as_sfc() |> st_as_sf()
step2 <- sx_filter(step1, box2, predicate = "intersects", output = "lazy")

# Collect final result
final <- sx_collect(step2)

# ------------------------------------------------------------------
# Example 3: Using pre-registered view names
# ------------------------------------------------------------------
sx_as_view(nc, "nc_counties")
sx_as_view(box, "filter_box")

# Filter using view names
result <- sx_filter("nc_counties", "filter_box", output = "sf")

# ------------------------------------------------------------------
# Example 4: Different predicates
# ------------------------------------------------------------------
# Filter with "within" predicate
```

```
within_result <- sx_filter(nc, box, predicate = "within", output = "sf")

# ---------------------------------------------------------------------
# Example 5: Interoperability Outputs
# ---------------------------------------------------------------------
# Export as geoarrow for Arrow/Parquet workflows
geoarrow_result <- sx_filter(nc, box, output = "geoarrow")

# Export as raw WKB for DuckDB/PostGIS import
raw_result <- sx_filter(nc, box, output = "raw")
```

`sx_geometry_column` *Get the name of the geometry column*

Description

Retrieves the name of the geometry column from an `sf` object, `sedonadb_dataframe`, or a registered SedonaDB view.

Usage

```
sx_geometry_column(x)
```

Arguments

`x`	A `sedonadb_dataframe`, `sf` object, or view name (character) in SedonaDB to inspect.

Value

Character. The name of the geometry column.

Examples

```
library(sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
sx_geometry_column(nc) # "geometry"

sdf <- sx_as_view(nc, "nc_view")
sx_geometry_column("nc_view") # "geometry"
```

`sx_interpolate_aw` *Areal-Weighted Interpolation using SedonaDB*

Description

Transfers attribute data from a source spatial layer to a target spatial layer based on the area of overlap between their geometries. All calculations are performed in SedonaDB for efficiency. Supports lazy evaluation returning `sedonadb_dataframe` objects.

Usage

```
sx_interpolate_aw(
  target,
  source,
  tid,
  sid,
  extensive = NULL,
  intensive = NULL,
  weight = "sum",
  output = NULL,
  view_name = NULL,
  keep_NA = TRUE,
  na.rm = FALSE,
  join_crs = NULL,
  verbosity = NULL,
  use_s2 = NULL,
  ...
)
```

Arguments

`target`	A `sedonadb_dataframe`, `sf` object, or view name (character) in SedonaDB representing destination geometries.
`source`	A `sedonadb_dataframe`, `sf` object, or view name (character) in SedonaDB containing data to interpolate.
`tid`	Character. Unique ID column name in `target`.
`sid`	Character. Unique ID column name in `source`.
`extensive`	Character vector. Columns in `source` to be treated as extensive (counts).
`intensive`	Character vector. Columns in `source` to be treated as intensive (rates).
`weight`	Character. Denominator for extensive variables: "sum" (default) or "total".
`output`	Character or NULL. Output type: `sedonadb_dataframe` (default), `sf`, `tibble`, `geoarrow`, or `raw`. If NULL, uses `getOption("sx.output_type", "sedonadb_dataframe")`. Output types: • `sedonadb_dataframe`: Lazy data frame (no collection).

- `sf`: Materialized sf object.
- `tibble`: Tibble without geometry.
- `geoarrow`: Tibble with `geoarrow_vctr` geometry (Arrow-native).
- `raw`: Tibble with geometry as raw WKB bytes (for database import).

`view_name` Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view.

Not all backends support named views. Check backend-specific documentation for availability.

`keep_NA` Logical. If TRUE, output includes all target features (LEFT JOIN).

`na.rm` Logical. If TRUE, source features with NA values are ignored.

`join_crs` Numeric or Character (optional). EPSG code or WKT for CRS transform during calc.

`verbosity` Character or NULL. Controls message output for this function call.

- `"quiet"`: Suppress all informational messages.
- `"info"`: Show standard progress and status messages.
- `"debug"`: Show additional diagnostic messages for troubleshooting.

If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.

`use_s2` Logical or NULL. Controls spherical geometry (S2) for this operation.

- `TRUE`: Use S2 spherical geometry (accurate for geographic coordinates).
- `FALSE`: Use planar geometry (faster, appropriate for projected CRS).
- `NULL` (default): Uses the global `sx_use_s2()` setting.

`...` Ignored. Used to catch and warn about unsupported sf arguments.

Details

Areal-weighted interpolation assumes uniform distribution of values within source polygons.

Coordinate Systems: Area calculations are sensitive to CRS. It is strongly recommended to use a projected CRS. Use the `join_crs` argument to project data on-the-fly during the interpolation.

Extensive vs. Intensive Variables:

- **Extensive** (counts, sums): Value is divided proportionally to area. Use `weight="sum"` (relative to target coverage) or `weight="total"` (relative to source area).
- **Intensive** (rates, densities): Value is averaged based on partial areas. Always uses intersection area weighting.

Value

An `sf` object, `sedonadb_dataframe`, or `tibble`.

See Also

`areal::aw_interpolate()` for reference implementation.

Examples

```
library(sf)

# 1. Prepare Data
# Load NC counties (source) and project to Albers (EPSG:5070)
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
nc <- st_transform(nc, 5070)
nc$sid <- seq_len(nrow(nc))

# Create a target grid
grid <- st_make_grid(nc, n = c(10, 5)) |> st_as_sf()
grid$tid <- seq_len(nrow(grid))

# -----------------------------------------------------------------
# Example 1: Using sf objects directly (most common use case)
# -----------------------------------------------------------------
# Extensive interpolation (total counts, e.g., births)
result_ext <- sx_interpolate_aw(
  target = grid, source = nc,
  tid = "tid", sid = "sid",
  extensive = "BIR74",
  weight = "total",
  output = "sf"
)

# Check mass preservation (should be ~1.0)
sum(result_ext$BIR74, na.rm = TRUE) / sum(nc$BIR74)

# Intensive interpolation (rates/densities)
result_int <- sx_interpolate_aw(
  target = grid, source = nc,
  tid = "tid", sid = "sid",
  intensive = "BIR74",
  output = "sf"
)

# -----------------------------------------------------------------
# Example 2: Using sedonadb_dataframe (lazy evaluation)
# -----------------------------------------------------------------
# First operation returns lazy result
lazy_result <- sx_interpolate_aw(
  target = grid, source = nc,
  tid = "tid", sid = "sid",
  extensive = c("BIR74", "BIR79"),
  output = "sedonadb_dataframe"
)

# Materialize when ready
final_sf <- sx_collect(lazy_result)

# -----------------------------------------------------------------
# Example 3: Using pre-registered SedonaDB view names
```

```
# ------------------------------------------------------------------
# Register data as views
sx_as_view(nc, "nc_counties")
sx_as_view(grid, "target_grid")

# Use view names as input
result_from_views <- sx_interpolate_aw(
  target = "target_grid", source = "nc_counties",
  tid = "tid", sid = "sid",
  extensive = "BIR74",
  output = "sf"
)

# Quick visualization
plot(result_ext["BIR74"], main = "Interpolated Births (1974)", border = NA)

# ------------------------------------------------------------------
# Example 4: Arrow ecosystem
# ------------------------------------------------------------------
# Export as geoarrow for zero-copy Parquet writing
geo_result <- sx_interpolate_aw(grid, nc, "tid", "sid", extensive = "BIR74", output = "geoarrow")
```

`sx_join` *Spatial join for sf objects using SedonaDB*

Description

Performs a spatial join between two `sf` objects or a spatial object and a registered SedonaDB view.

Usage

```
sx_join(
  x,
  y,
  join = "intersects",
  distance = NULL,
  left = TRUE,
  largest = FALSE,
  output = NULL,
  view_name = NULL,
  overwrite = FALSE,
  verbosity = NULL,
  use_s2 = NULL,
  ...
)
```

Arguments

x	A sedonadb_dataframe, sf object, or view name (character) in SedonaDB representing the left-hand side of the join.
y	A sedonadb_dataframe, sf object, or view name (character) in SedonaDB representing the right-hand side of the join.
join	Character. Spatial predicate to use. One of: intersects (default), contains, within, touches, crosses, overlaps, covers, covered_by, disjoint, equals, contains_properly, or dwithin.
distance	Numeric (optional). Distance threshold for the dwithin predicate. Required when predicate = "dwithin".
left	Logical. If TRUE (default), performs a left join returning all features from x with NA values for non-matching y attributes. If FALSE, performs an inner join returning only features from x that match y. Matches sf::st_join behavior.
largest	Logical. If TRUE, returns only the y feature with the largest overlap for each x feature. If FALSE (default), returns all matching y features. Overlap is measured by area for polygons, length for lines, and treated as equal (1.0) for points. Matches sf::st_join behavior.
output	Character or NULL. Output type: sedonadb_dataframe (default), sf, tibble, geoarrow, or raw. If NULL, uses getOption("sx.output_type", "sedonadb_dataframe"). Output types: • sedonadb_dataframe: Lazy data frame (no collection). • sf: Materialized sf object. • tibble: Tibble without geometry. • geoarrow: Tibble with geoarrow_vctr geometry (Arrow-native). • raw: Tibble with geometry as raw WKB bytes (for database import).
view_name	Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view. Not all backends support named views. Check backend-specific documentation for availability.
overwrite	Logical. If TRUE, overwrites any existing view with the same name. Default is FALSE.
verbosity	Character or NULL. Controls message output for this function call. • "quiet": Suppress all informational messages. • "info": Show standard progress and status messages. • "debug": Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.
use_s2	Logical or NULL. Controls spherical geometry (S2) for this operation. • TRUE: Use S2 spherical geometry (accurate for geographic coordinates). • FALSE: Use planar geometry (faster, appropriate for projected CRS). • NULL (default): Uses the global sx_use_s2() setting.
...	Ignored. Used to catch and warn about unsupported sf arguments.

Value

Depends on `output`. Defaults to `sedonadb_dataframe`.

Examples

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

# Create points to join against counties
set.seed(42)
pts <- st_sample(st_union(nc), 50) |> st_as_sf()
pts$point_id <- seq_len(nrow(pts))

# ----------------------------------------------------------------------
# Example 1: Using sf objects (most common)
# ----------------------------------------------------------------------
# Join point attributes to the county that contains them
joined <- sx_join(pts, nc, join = "within")

# ----------------------------------------------------------------------
# Example 2: Using pre-registered view names
# ----------------------------------------------------------------------
sx_as_view(nc, "nc_counties")
sx_as_view(pts, "sample_points")

# Join using view names
joined_from_views <- sx_join("sample_points", "nc_counties", join = "within")

# ----------------------------------------------------------------------
# Example 3: Create a persistent view (named output)
# ----------------------------------------------------------------------
# Instead of returning sf, create a named view for later use
sx_join(pts, nc, join = "intersects", view_name = "points_with_county")

# The view can now be used in other operations
result <- sx_collect("points_with_county")

# ----------------------------------------------------------------------
# Example 4: Interoperability
# ----------------------------------------------------------------------
# Export as raw WKB for database import
wkb_result <- sx_join(pts, nc, output = "raw")
```

`sx_layers` *List layers in a spatial file*

Description

Lists all layers available in a spatial file (e.g., for GeoPackage, FileGDB, or PostGIS).

Usage

```
sx_layers(path, verbosity = NULL)
```

Arguments

path	Character string. Path to the file (local, HTTP, or S3) to inspect.
verbosity	Character or NULL. Controls message output for this function call. • "quiet": Suppress all informational messages. • "info": Show standard progress and status messages. • "debug": Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.

Value

A data frame with layer information, typically including layer_name, geometry_type, and feature_count.

Examples

```
# Path to a multi-layer GPKG
gpkg_path <- system.file("spatial/countries.geojson", package = "sx") # Example
# sx_layers(gpkg_path)
```

---

sx_limit_duckdb_conn *Manually applies sx resource limits (threads, memory) to a given DuckDB connection.*

Description

Manually applies sx resource limits (threads, memory) to a given DuckDB connection.

Usage

```
sx_limit_duckdb_conn(conn, threads = NULL, memory_limit_gb = NULL)
```

Arguments

conn	A DuckDB connection object (from DBI::dbConnect).
threads	Integer. Number of threads to use. If NULL (default), uses the global sx.duckdb.threads option. Providing a value pins the connection.
memory_limit_gb	Numeric or character. Memory limit to use. If NULL (default), uses the global sx.duckdb.memory_limit_gb option. Providing a value pins the connection.

Details

When `threads` or `memory_limit_gb` are explicitly provided, the connection is "pinned" in an internal registry. Pinned connections are **immune** to global policy enforcement (see `sx_options(duckdb_enforcement_mode = "all")`), ensuring that manual engineering decisions are respected.

To unpin a connection and return it to global policy management, call this function without any limit arguments.

Value

Invisibly returns the connection.

`sx_list` *List registered tables and views*

Description

Returns a list of all tables and views currently registered in the SedonaDB context.

Usage

```
sx_list(verbosity = NULL)
```

Arguments

`verbosity`	Character or NULL. Controls message output for this function call. • `"quiet"`: Suppress all informational messages. • `"info"`: Show standard progress and status messages. • `"debug"`: Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.

Value

A tibble containing catalog information (`table_name`, `table_type`, etc.).

`sx_options` *Get or set global sx options*

Description

Get or set global sx options

Usage

```
sx_options(
  output_type = NULL,
  verbosity = NULL,
  threads = NULL,
  memory_limit_gb = NULL,
  duckdb_threads = NULL,
  duckdb_memory_limit_gb = NULL,
  duckdb_enforcement_mode = NULL
)
```

Arguments

`output_type` Character string. Controls the default return type for spatial operations. Must be one of:

- `sedonadb_dataframe` (default): Lazy data frame (no collection).
- `sf`: Materialized sf object.
- `tibble`: Tibble without geometry.
- `geoarrow`: Tibble with `geoarrow_vctr` geometry (Arrow-native).
- `raw`: Tibble with geometry as raw WKB bytes (for database import).

If `NULL` (the default), the existing option is not changed.

`verbosity` Character or NULL. Controls message output level:

- `"quiet"`: Suppress all informational messages.
- `"info"` (default): Show standard progress and status messages.
- `"debug"`: Show additional diagnostic messages for troubleshooting.

`threads` Positive integer. Number of threads to use for SedonaDB parallel operations.

`memory_limit_gb`
Positive numeric. Memory limit in GB for SedonaDB backend operations. Note: For SedonaDB/DataFusion, this setting is applied at the runtime level and may not be visible via SQL `SHOW ALL`.

`duckdb_threads` Positive integer or "Auto". Threads for DuckDB operations.

`duckdb_memory_limit_gb`
Positive numeric or "Auto". Memory limit for DuckDB (e.g., 4 or "4GB").

`duckdb_enforcement_mode`
Character. Controls resource limit enforcement scope:

- `"internal"` (default): Limits apply only to internal `sx` connections.
- `"all"`: Limits apply to **all** observable DuckDB connections.

Note on Policy Immunity: Connections manually configured via `sx_limit_duckdb_conn()` (with explicit limits) are "pinned" in an internal registry. Pinned connections are **immune** to global policy enforcement, ensuring that manual engineering decisions are respected.

If `NULL` (the default), the existing option is not changed.

Value

Invisibly returns a list containing the currently set options when setting. Returns the list visibly when called without arguments (getter mode).

Examples

```
## Not run:
# --- Output Type Options ---
# Set default output to sf (materialized)
sx_options(output_type = "sf")

# --- Independent Resource Config ---
sx_options(
  threads = 8,                 # SedonaDB (Heavy compute)
  duckdb_threads = 2,          # DuckDB (I/O only)
  duckdb_memory_limit_gb = 4   # DuckDB specific limit
)

# --- View Current Configuration ---
sx_options()

## End(Not run)
```

---

sx_read *Read spatial data into SedonaDB*

Description

This function reads spatial files and transfers them to SedonaDB. Parquet files (local, HTTP, S3) are read natively by SedonaDB. Other formats (Shapefile, GeoJSON, GPKG) are read via DuckDB and streamed via Arrow.

Usage

```
sx_read(
  path,
  data_reader = "auto",
  query = NULL,
  view_name = NULL,
```

```
  target = "sedonadb",
  options = NULL,
  shp_encoding = NULL,
  layer = NULL,
  spatial_filter = NULL,
  open_options = NULL,
  allowed_drivers = NULL,
  hive_partitioning = NULL,
  union_by_name = NULL,
  max_batch_size = NULL,
  verbosity = NULL,
  ...
)
```

Arguments

path	Character string. Path to the file (local, HTTP, or S3) to read. Can also be a table name if using the DuckDB reader with a custom conn.
data_reader	Character string specifying which data reader to use. Options: • "auto" (default): Auto-detect based on file type • "sedonadb": Native SedonaDB reader (parquet only) • "duckdb": Use DuckDB spatial reader (shp, gpkg, geojson, etc.) Most users should use "auto".
query	Optional SQL query (for DuckDB reader only). If NULL, reads all data. Use %s placeholder for the path if needed.
view_name	Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view. Not all backends support named views. Check backend-specific documentation for availability.
target	Target engine for loading. Currently only "sedonadb" is supported.
options	Named list of options for SedonaDB parquet reader (e.g., S3 credentials). Example: list("aws.region" = "us-west-2", "aws.skip_signature" = TRUE)
shp_encoding	Character encoding for Shapefile attribute data (e.g., "UTF-8", "CP1252"). Only used when reading Shapefiles via DuckDB. Useful for non-ASCII characters.
layer	Layer name to read (for multi-layer formats like GPKG).
spatial_filter	WKT string or sf/sfc object defining a spatial filter bounding box. Only rows intersecting this box will be read.
open_options	Character vector of driver-specific open options for GDAL (e.g., c("HEADERS=FORCE") for CSV). Passed to ST_Read.
allowed_drivers	Character vector of GDAL driver names to restrict reading to.
hive_partitioning	Logical. For partitioned Parquet directories, if TRUE, interprets the directory structure as Hive-style partitioning.

union_by_name	Logical. For multi-file Parquet reads, if TRUE, unifies columns by name across files (handles schema variations).
max_batch_size	Integer. Maximum batch size for GDAL reads via ST_Read.
verbosity	Character or NULL. Controls message output for this function call. • "quiet": Suppress all informational messages. • "info": Show standard progress and status messages. • "debug": Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.
...	Additional arguments passed to the data reader (e.g., conn for DuckDB).

Value

A sedonadb_dataframe.

Examples

```
## Not run:
# Auto-detect: parquet uses SedonaDB, shp uses DuckDB
sdf <- sx_read("path/to/file.parquet")
sdf <- sx_read("path/to/file.shp")

# S3 parquet with options
sdf <- sx_read(
  "s3://bucket/path/file.parquet",
  options = list("aws.region" = "us-west-2")
)

# HTTP parquet
sdf <- sx_read("https://example.com/data.parquet")

# Read shapefile with explicit encoding
df <- sx_read("data.shp", shp_encoding = "CP1252")

# Read specific layer from GPKG
df <- sx_read("data.gpkg", layer = "counties")

# Read with driver-specific open options
df <- sx_read("data.csv", open_options = c("HEADERS=FORCE"))

# Read partitioned parquet with Hive partitioning
df <- sx_read("data_partitioned/", hive_partitioning = TRUE)

# Read from existing DuckDB table
df <- sx_read("my_table", conn = my_duckdb_conn)

## End(Not run)
```

`sx_simplify` *Simplify geometries*

Description

Simplify the geometry by removing points (vertices) that do not significantly contribute to the shape. Uses the Ramer-Douglas-Peucker algorithm.

Usage

```
sx_simplify(
  x,
  dTolerance,
  preserveTopology = FALSE,
  output = NULL,
  view_name = NULL,
  verbosity = NULL,
  ...
)
```

Arguments

`x`	Input object (sf, sedonadb_dataframe, or character view name).
`dTolerance`	Numeric. The tolerance distance for simplification. Vertices closer than this distance to the simplified line are removed.
`preserveTopology`	Logical. If `TRUE`, attempts to preserve the topology of the geometry (e.g., preventing self-intersections). (Not yet supported/verified for SedonaDB backend, defaults to FALSE. Warning issued if TRUE).
`output`	Character or NULL. Output type: `sedonadb_dataframe` (default), `sf`, `tibble`, `geoarrow`, or `raw`. If NULL, uses `getOption("sx.output_type", "sedonadb_dataframe")`. Output types: • `sedonadb_dataframe`: Lazy data frame (no collection). • `sf`: Materialized sf object. • `tibble`: Tibble without geometry. • `geoarrow`: Tibble with `geoarrow_vctr` geometry (Arrow-native). • `raw`: Tibble with geometry as raw WKB bytes (for database import).
`view_name`	Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view. Not all backends support named views. Check backend-specific documentation for availability.
`verbosity`	Character or NULL. Controls message output for this function call. • `"quiet"`: Suppress all informational messages. • `"info"`: Show standard progress and status messages.

- "debug": Show additional diagnostic messages for troubleshooting.

If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.

... Ignored. Used to catch and warn about unsupported sf arguments.

Value

Result (type depends on output)

---

sx_sitrep *Report sx configuration status*

Description

Displays useful information about the current configuration, including global options and the status of the SedonaDB context.

Usage

```
sx_sitrep(verbosity = NULL)
```

Arguments

verbosity Character or NULL. Controls message output for this function call.

- "quiet": Suppress all informational messages.
- "info": Show standard progress and status messages.
- "debug": Show additional diagnostic messages for troubleshooting.

If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.

Value

Invisibly returns a list with the current status configuration.

Examples

```
sx_sitrep()
```

sx_sql *Execute SQL with R object interpolation*

Description

Executes a SQL query against SedonaDB, allowing R `sedonadb_dataframe` objects or `sf` objects to be used directly in the SQL string via `{}` interpolation.

Supports piping: `df |> sx_sql("SELECT * FROM {.}")`.

Usage

```
sx_sql(..., .envir = parent.frame(), verbosity = NULL)
```

Arguments

`...`	SQL string parts (passed to `glue::glue`). The first argument can optionally be a data object (context) which is ignored in string construction but available for interpolation as `{.}`.
`.envir`	Environment to evaluate expressions in. Defaults to `parent.frame()`.
`verbosity`	Character or NULL. Controls message output for this function call. • `"quiet"`: Suppress all informational messages. • `"info"`: Show standard progress and status messages. • `"debug"`: Show additional diagnostic messages for troubleshooting. If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.

Value

A `sedonadb_dataframe`.

Examples

```
## Not run:
nc <- sx_read("shape/nc.shp")

# Reference object by name
sx_sql("SELECT * FROM {nc} WHERE AREA > 0.1")

# Reference via pipe
nc |> sx_sql("SELECT * FROM {.} WHERE AREA > 0.1")

## End(Not run)
```

`sx_transform` *Transform coordinate reference system*

Description

Transforms the coordinates of the geometry column to a new Coordinate Reference System (CRS). Calculations are performed in SedonaDB using `ST_Transform`.

Usage

```
sx_transform(
  x,
  crs,
  src_crs = NULL,
  output = NULL,
  view_name = NULL,
  verbosity = NULL,
  ...
)
```

Arguments

`x`	A `sedonadb_dataframe`, `sf` object, or view name (character) in SedonaDB to transform.
`crs`	Target Coordinate Reference System. Can be: • An integer/numeric EPSG code (e.g., `5070`). • A character string (e.g., `"EPSG:5070"`). • An `sf::crs` object. • An `sf` object (to extract CRS from).
`src_crs`	Source Coordinate Reference System (optional). If `NULL` (default), the CRS is inferred from the input `x` using `sx_crs()`. If the input has no CRS, this argument is required.
`output`	Character or NULL. Output type: `sedonadb_dataframe` (default), `sf`, `tibble`, `geoarrow`, or `raw`. If NULL, uses `getOption("sx.output_type", "sedonadb_dataframe")`. Output types: • `sedonadb_dataframe`: Lazy data frame (no collection). • `sf`: Materialized sf object. • `tibble`: Tibble without geometry. • `geoarrow`: Tibble with `geoarrow_vctr` geometry (Arrow-native). • `raw`: Tibble with geometry as raw WKB bytes (for database import).
`view_name`	Character (optional). Name to register the result as a persistent view in the active backend. If NULL (default), returns the result directly without creating a view. Not all backends support named views. Check backend-specific documentation for availability.

verbosity — Character or NULL. Controls message output for this function call.

- "quiet": Suppress all informational messages.
- "info": Show standard progress and status messages.
- "debug": Show additional diagnostic messages for troubleshooting.

If NULL (the default), uses the global sx.verbosity option. See sx_options() for persistent configuration.

... — Ignored. Used to catch sf-specific arguments.

Value

Result (type depends on output)

See Also

sf::st_transform()

Examples

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

# Transform to EPSG:5070 (Albers)
res <- sx_transform(nc, 5070, output = "sf")

# Register as view and transform
sx_as_view(nc, "nc_raw")
sx_transform("nc_raw", "EPSG:3857")
```

---

sx_use_s2 *Set or get the global S2 usage preference for sx*

Description

Explicitly controls whether sx functions should use spherical geometry (S2) logic by default.

Usage

```
sx_use_s2(use)
```

Arguments

use — Logical. TRUE to enable S2 by default, FALSE to disable. If missing, returns the current global setting.

Details

This setting acts as a global default for sx functions. It can comprise:

- **Global Default**: Unset (NULL). sx defaults to TRUE (Geometric safety) if backend supports it.
- **User Override**: Set via this function.
- **Per-Call Override**: Arguments use_s2 in sx_* functions take highest precedence.

Value

Logical. The current setting (invisibly if use is provided).

---

sx_write *Write spatial data to disk*

Description

Writes spatial data to disk. Parquet files use SedonaDB's native GeoParquet writer. Other formats (GeoJSON, Shapefile, GPKG) use DuckDB's GDAL-based COPY.

Usage

```
sx_write(
  data,
  path,
  gdal_driver = NULL,
  overwrite = FALSE,
  crs = NULL,
  compression = NULL,
  options = list(),
  verbosity = NULL,
  ...
)
```

Arguments

data	A sedonadb_dataframe, sf object, or view name (character) in SedonaDB. Can also be a character string specifying the input file path for reading operations to write.
path	Character string. Path to the file (local, HTTP, or S3) to write to.
gdal_driver	GDAL driver name for writing spatial formats. If NULL (default), the driver is auto-detected from the file extension: • .geojson, .json → "GeoJSON" • .shp → "ESRI Shapefile" • .gpkg → "GPKG"

- `.fgb` → "FlatGeobuf"

For non-standard extensions, specify the driver explicitly. Use `"parquet"` extension for native GeoParquet output.

`overwrite` Logical. If `TRUE`, overwrites any existing view with the same name. Default is `FALSE`.

`crs` Output CRS (e.g., "EPSG:4326"). Passed to GDAL as `SRS` option. If `NULL`, auto-detects from input data.

`compression` Compression codec for Parquet files. Common options: `"zstd"`, `"snappy"`, `"gzip"`, `"lz4"`, `"uncompressed"`. Default `NULL` uses the backend's default (SedonaDB: uncompressed via native writer, DuckDB: snappy). **Note**: Specifying compression routes parquet output through DuckDB instead of SedonaDB's native GeoParquet writer.

`options` Named list of additional options:

- For parquet (via SedonaDB native writer, when no compression):
  - `partition_by`: Character vector of column names for Hive-style partitioning
  - `sort_by`: Character vector of column names to sort by (ascending)
  - `single_file_output`: TRUE/FALSE to force single file vs directory output
  - `geoparquet_version`: "1.0" (default) or "1.1" for bbox columns
  - `overwrite_bbox_columns`: If TRUE, overwrites existing bbox columns
- For parquet (via DuckDB, when compression specified):
  - `ROW_GROUP_SIZE`: Integer, row group size
- For GDAL: `LAYER_CREATION_OPTIONS`

`verbosity` Character or NULL. Controls message output for this function call.

- `"quiet"`: Suppress all informational messages.
- `"info"`: Show standard progress and status messages.
- `"debug"`: Show additional diagnostic messages for troubleshooting.

If NULL (the default), uses the global `sx.verbosity` option. See `sx_options()` for persistent configuration.

`...` Additional arguments (reserved for future use).

Value

The `path` invisibly.

Examples

```
## Not run:
# Read nc data
sdf <- sx_read(system.file("shape/nc.shp", package = "sf"))

# Write to GeoJSON (auto-detected)
sx_write(sdf, "nc.geojson")
```

```
# Write to GeoParquet (native SedonaDB writer)
sx_write(sdf, "nc.parquet")

# Write to compressed parquet (via DuckDB)
sx_write(sdf, "nc_compressed.parquet", compression = "zstd")

# Write to Shapefile with explicit driver
sx_write(sdf, "nc.shp", gdal_driver = "ESRI Shapefile")

# Overwrite existing file
sx_write(sdf, "nc.gpkg", overwrite = TRUE)

# CRS override
sx_write(sdf, "nc_3857.geojson", crs = "EPSG:3857")

## End(Not run)
```

Index