

# Package ‘apache.sedona’

June 8, 2025

**Type** Package

**Title** R Interface for Apache Sedona

**Version** 1.7.2

**Maintainer** Apache Sedona <private@sedona.apache.org>

**Description** R interface for 'Apache Sedona' based on 'sparklyr'  
(<<https://sedona.apache.org>>).

**License** Apache License 2.0

**URL** <https://github.com/apache/sedona/>, <https://sedona.apache.org/>

**BugReports** <https://github.com/apache/sedona/issues>

**Depends** R (>= 3.2)

**Imports** rlang, sparklyr (>= 1.3), dbplyr (>= 1.1.0), cli, lifecycle

**Suggests** dplyr (>= 0.7.2), knitr, rmarkdown

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**SystemRequirements** 'Apache Spark' 3.x

**NeedsCompilation** no

**Author** Apache Sedona [aut, cre],  
Jia Yu [ctb, cph],  
Yitao Li [aut, cph] (ORCID: <<https://orcid.org/0000-0002-1261-905X>>),  
The Apache Software Foundation [cph],  
RStudio [cph]

**Repository** CRAN

**Date/Publication** 2025-06-08 11:20:02 UTC

## Contents

|                                |   |
|--------------------------------|---|
| approx_count . . . . .         | 2 |
| crs_transform . . . . .        | 3 |
| minimum_bounding_box . . . . . | 4 |

|                                              |           |
|----------------------------------------------|-----------|
| new_bounding_box . . . . .                   | 5         |
| sdf_register.spatial_rdd . . . . .           | 6         |
| sedona_apply_spatial_partitioner . . . . .   | 7         |
| sedona_build_index . . . . .                 | 8         |
| sedona_knn_query . . . . .                   | 9         |
| sedona_range_query . . . . .                 | 11        |
| sedona_read_dsv_to_typed_rdd . . . . .       | 12        |
| sedona_read_geojson . . . . .                | 14        |
| sedona_read_shapefile_to_typed_rdd . . . . . | 16        |
| sedona_render_choropleth_map . . . . .       | 17        |
| sedona_render_heatmap . . . . .              | 19        |
| sedona_render_scatter_plot . . . . .         | 21        |
| sedona_save_spatial_rdd . . . . .            | 23        |
| sedona_spatial_join . . . . .                | 24        |
| sedona_spatial_join_count_by_key . . . . .   | 25        |
| sedona_write_wkb . . . . .                   | 27        |
| spark_read_shapefile . . . . .               | 28        |
| spark_write_geojson . . . . .                | 30        |
| to_spatial_rdd . . . . .                     | 31        |
| <b>Index</b>                                 | <b>33</b> |

---

|              |                                                                           |
|--------------|---------------------------------------------------------------------------|
| approx_count | <i>Find the approximate total number of records within a Spatial RDD.</i> |
|--------------|---------------------------------------------------------------------------|

---

**Description**

Given a Sedona spatial RDD, find the (possibly approximated) number of total records within it.

**Usage**

approx\_count(x)

**Arguments**

x                      A Sedona spatial RDD.

**Value**

Approximate number of records within the SpatialRDD.

**See Also**

Other Spatial RDD aggregation routine: [minimum\\_bounding\\_box\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_shapefile_to_typed_rdd(
    sc,
    location = input_location, type = "polygon"
  )
  approx_cnt <- approx_count(rdd)
}
```

---

|               |                                      |
|---------------|--------------------------------------|
| crs_transform | <i>Perform a CRS transformation.</i> |
|---------------|--------------------------------------|

---

**Description**

Transform data within a spatial RDD from one coordinate reference system to another. This uses the lon/lat order since v1.5.0. Before, it used lat/lon

**Usage**

```
crs_transform(x, src_epsg_crs_code, dst_epsg_crs_code, strict = FALSE)
```

**Arguments**

|                   |                                                                                      |
|-------------------|--------------------------------------------------------------------------------------|
| x                 | The spatial RDD to be processed.                                                     |
| src_epsg_crs_code | Coordinate reference system to transform from (e.g., "epsg:4326", "epsg:3857", etc). |
| dst_epsg_crs_code | Coordinate reference system to transform to. (e.g., "epsg:4326", "epsg:3857", etc).  |
| strict            | If FALSE (default), then ignore the "Bursa-Wolf Parameters Required" error.          |

**Value**

The transformed SpatialRDD.

### Examples

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_geojson_to_typed_rdd(
    sc,
    location = input_location, type = "polygon"
  )
  crs_transform(
    rdd,
    src_epsg_crs_code = "epsg:4326", dst_epsg_crs_code = "epsg:3857"
  )
}
```

---

|                      |                                                     |
|----------------------|-----------------------------------------------------|
| minimum_bounding_box | <i>Find the minimal bounding box of a geometry.</i> |
|----------------------|-----------------------------------------------------|

---

### Description

Given a Sedona spatial RDD, find the axis-aligned minimal bounding box of the geometry represented by the RDD.

### Usage

```
minimum_bounding_box(x)
```

### Arguments

|   |                       |
|---|-----------------------|
| x | A Sedona spatial RDD. |
|---|-----------------------|

### Value

A minimum bounding box object.

### See Also

Other Spatial RDD aggregation routine: [approx\\_count\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_shapefile_to_typed_rdd(
    sc,
    location = input_location, type = "polygon"
  )
  boundary <- minimum_bounding_box(rdd)
}
```

---

|                  |                                         |
|------------------|-----------------------------------------|
| new_bounding_box | <i>Construct a bounding box object.</i> |
|------------------|-----------------------------------------|

---

**Description**

Construct a axis-aligned rectangular bounding box object.

**Usage**

```
new_bounding_box(sc, min_x = -Inf, max_x = Inf, min_y = -Inf, max_y = Inf)
```

**Arguments**

|       |                                                      |
|-------|------------------------------------------------------|
| sc    | The Spark connection.                                |
| min_x | Minimum x-value of the bounding box, can be +/- Inf. |
| max_x | Maximum x-value of the bounding box, can be +/- Inf. |
| min_y | Minimum y-value of the bounding box, can be +/- Inf. |
| max_y | Maximum y-value of the bounding box, can be +/- Inf. |

**Value**

A bounding box object.

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")
bb <- new_bounding_box(sc, -1, 1, -1, 1)
```

---

sdf\_register.spatial\_rdd

*Import data from a spatial RDD into a Spark Dataframe.*


---

## Description

Import data from a spatial RDD (possibly with non-spatial attributes) into a Spark Dataframe.

- sdf\_register: method for sparklyr's sdf\_register to handle Spatial RDD
- as.spark.dataframe: lower level function with more fine-grained control on non-spatial columns

## Usage

```
## S3 method for class 'spatial_rdd'
sdf_register(x, name = NULL)

as.spark.dataframe(x, non_spatial_cols = NULL, name = NULL)
```

## Arguments

|                  |                                                                                                                                                                                   |
|------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x                | A spatial RDD.                                                                                                                                                                    |
| name             | Name to assign to the resulting Spark temporary view. If unspecified, then a random name will be assigned.                                                                        |
| non_spatial_cols | Column names for non-spatial attributes in the resulting Spark Dataframe. By default (NULL) it will import all field names if that property exists, in particular for shapefiles. |

## Value

A Spark Dataframe containing the imported spatial data.

## Examples

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_geojson_to_typed_rdd(
    sc,
    location = input_location,
    type = "polygon"
  )
  sdf <- sdf_register(rdd)
```

```

input_location <- "/dev/null" # replace it with the path to your input file
rdd <- sedona_read_dsv_to_typed_rdd(
  sc,
  location = input_location,
  delimiter = ",",
  type = "point",
  first_spatial_col_index = 1L,
  repartition = 5
)
sdf <- as.spark.dataframe(rdd, non_spatial_cols = c("attr1", "attr2"))
}

```

---

sedona\_apply\_spatial\_partitioner

*Apply a spatial partitioner to a Sedona spatial RDD.*

---

## Description

Given a Sedona spatial RDD, partition its content using a spatial partitioner.

## Usage

```

sedona_apply_spatial_partitioner(
  rdd,
  partitioner = c("quadtree", "kdmtree"),
  max_levels = NULL
)

```

## Arguments

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rdd         | The spatial RDD to be partitioned.                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| partitioner | The name of a grid type to use (currently "quadtree" and "kdmtree" are supported) or an <code>org.apache.sedona.core.spatialPartitioning.SpatialPartitioner</code> JVM object. The latter option is only relevant for advanced use cases involving a custom spatial partitioner.                                                                                                                                                                                           |
| max_levels  | Maximum number of levels in the partitioning tree data structure. If NULL (default), then use the current number of partitions within rdd as maximum number of levels. Specifying max_levels is unsupported for use cases involving a custom spatial partitioner because in these scenarios the partitioner object already has its own maximum number of levels set and there is no well-defined way to override this existing setting in the partitioning data structure. |

## Value

A spatially partitioned SpatialRDD.

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_dsv_to_typed_rdd(
    sc,
    location = input_location,
    delimiter = ",",
    type = "point",
    first_spatial_col_index = 1L
  )
  sedona_apply_spatial_partitioner(rdd, partitioner = "kdbtree")
}
```

---

|                    |                                                |
|--------------------|------------------------------------------------|
| sedona_build_index | <i>Build an index on a Sedona spatial RDD.</i> |
|--------------------|------------------------------------------------|

---

**Description**

Given a Sedona spatial RDD, build the type of index specified on each of its partition(s).

**Usage**

```
sedona_build_index(
  rdd,
  type = c("quadtree", "rtree"),
  index_spatial_partitions = TRUE
)
```

**Arguments**

|                          |                                                                                                                                                                                                                                                                                                                                              |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rdd                      | The spatial RDD to be indexed.                                                                                                                                                                                                                                                                                                               |
| type                     | The type of index to build. Currently "quadtree" and "rtree" are supported.                                                                                                                                                                                                                                                                  |
| index_spatial_partitions | If the RDD is already partitioned using a spatial partitioner, then index each spatial partition within the RDD instead of partitions within the raw RDD associated with the underlying spatial data source. Default: TRUE. Notice this option is irrelevant if the input RDD has not been partitioned using with a spatial partitioner yet. |

**Value**

A spatial index object.



**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_shapefile_to_typed_rdd(
    sc,
    location = input_location,
    type = "polygon"
  )
  sedona_build_index(rdd, type = "rtree")
}
```

---

sedona\_knn\_query

*Query the k nearest spatial objects.*


---

**Description**

Given a spatial RDD, a query object x, and an integer k, find the k nearest spatial objects within the RDD from x (distance between x and another geometrical object will be measured by the minimum possible length of any line segment connecting those 2 objects).

**Usage**

```
sedona_knn_query(
  rdd,
  x,
  k,
  index_type = c("quadtree", "rtree"),
  result_type = c("rdd", "sdf", "raw")
)
```

**Arguments**

|            |                                                                                                                                                                  |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rdd        | A Sedona spatial RDD.                                                                                                                                            |
| x          | The query object.                                                                                                                                                |
| k          | Number of nearest spatail objects to return.                                                                                                                     |
| index_type | Index to use to facilitate the KNN query. If NULL, then do not build any additional spatial index on top of x. Supported index types are "quadtree" and "rtree". |

**result\_type**      Type of result to return. If "rdd" (default), then the k nearest objects will be returned in a Sedona spatial RDD. If "sdf", then a Spark dataframe containing the k nearest objects will be returned. If "raw", then a list of k nearest objects will be returned. Each element within this list will be a JVM object of type `org.locationtech.jts.geom.Geometry`.

## Value

The KNN query result.

## See Also

Other Sedona spatial query: [sedona\\_range\\_query\(\)](#)

## Examples

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  knn_query_pt_x <- -84.01
  knn_query_pt_y <- 34.01
  knn_query_pt_tbl <- sdf_sql(
    sc,
    sprintf(
      "SELECT ST_GeomFromText(\"POINT(%f %f)\") AS `pt`",
      knn_query_pt_x,
      knn_query_pt_y
    )
  ) %>%
  collect()
  knn_query_pt <- knn_query_pt_tbl$pt[[1]]
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_geojson_to_typed_rdd(
    sc,
    location = input_location,
    type = "polygon"
  )
  knn_result_sdf <- sedona_knn_query(
    rdd,
    x = knn_query_pt, k = 3, index_type = "rtree", result_type = "sdf"
  )
}
```

---

|                    |                               |
|--------------------|-------------------------------|
| sedona_range_query | <i>Execute a range query.</i> |
|--------------------|-------------------------------|

---

**Description**

Given a spatial RDD and a query object x, find all spatial objects within the RDD that are covered by x or intersect x.

**Usage**

```
sedona_range_query(
  rdd,
  x,
  query_type = c("cover", "intersect"),
  index_type = c("quadtree", "rtree"),
  result_type = c("rdd", "sdf", "raw")
)
```

**Arguments**

|             |                                                                                                                                                                                                                                                                                                                                                                    |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rdd         | A Sedona spatial RDD.                                                                                                                                                                                                                                                                                                                                              |
| x           | The query object.                                                                                                                                                                                                                                                                                                                                                  |
| query_type  | Type of spatial relationship involved in the query. Currently "cover" and "intersect" are supported.                                                                                                                                                                                                                                                               |
| index_type  | Index to use to facilitate the KNN query. If NULL, then do not build any additional spatial index on top of x. Supported index types are "quadtree" and "rtree".                                                                                                                                                                                                   |
| result_type | Type of result to return. If "rdd" (default), then the k nearest objects will be returned in a Sedona spatial RDD. If "sdf", then a Spark dataframe containing the k nearest objects will be returned. If "raw", then a list of k nearest objects will be returned. Each element within this list will be a JVM object of type org.locationtech.jts.geom.Geometry. |

**Value**

The range query result.

**See Also**

Other Sedona spatial query: [sedona\\_knn\\_query\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")
```

```

if (!inherits(sc, "test_connection")) {
  range_query_min_x <- -87
  range_query_max_x <- -50
  range_query_min_y <- 34
  range_query_max_y <- 54
  geom_factory <- invoke_new(
    sc,
    "org.locationtech.jts.geom.GeometryFactory"
  )
  range_query_polygon <- invoke_new(
    sc,
    "org.locationtech.jts.geom.Envelope",
    range_query_min_x,
    range_query_max_x,
    range_query_min_y,
    range_query_max_y
  ) %>%
    invoke(geom_factory, "toGeometry", .)
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_geojson_to_typed_rdd(
    sc,
    location = input_location,
    type = "polygon"
  )
  range_query_result_sdf <- sedona_range_query(
    rdd,
    x = range_query_polygon,
    query_type = "intersect",
    index_type = "rtree",
    result_type = "sdf"
  )
}

```

---

sedona\_read\_dsv\_to\_typed\_rdd

*Create a typed SpatialRDD from a delimiter-separated values data source.*

---

## Description

Create a typed SpatialRDD (namely, a PointRDD, a PolygonRDD, or a LineStringRDD) from a data source containing delimiter-separated values. The data source can contain spatial attributes (e.g., longitude and latitude) and other attributes. Currently only inputs with spatial attributes occupying a contiguous range of columns (i.e., [first\_spatial\_col\_index, last\_spatial\_col\_index]) are supported.

## Usage

```
sedona_read_dsv_to_typed_rdd(
```

```

    sc,
    location,
    delimiter = c(",", "\t", "?", "'", "\"", "_", "-", "%", "~", "|", ";"),
    type = c("point", "polygon", "linestring"),
    first_spatial_col_index = 0L,
    last_spatial_col_index = NULL,
    has_non_spatial_attrs = TRUE,
    storage_level = "MEMORY_ONLY",
    repartition = 1L
  )

```

### Arguments

|                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|--------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>sc</code>                      | A <code>spark_connection</code> .                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>location</code>                | Location of the data source.                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>delimiter</code>               | Delimiter within each record. Must be one of <code>','</code> , <code>'\t'</code> , <code>'?'</code> , <code>'\''</code> , <code>'\"'</code> , <code>'_'</code> , <code>'-'</code> , <code>'%'</code> , <code>'~'</code> , <code>' '</code> , <code>';</code> .                                                                                                                                                                                               |
| <code>type</code>                    | Type of the SpatialRDD (must be one of "point", "polygon", or "linestring").                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>first_spatial_col_index</code> | Zero-based index of the left-most column containing spatial attributes (default: 0).                                                                                                                                                                                                                                                                                                                                                                          |
| <code>last_spatial_col_index</code>  | Zero-based index of the right-most column containing spatial attributes (default: NULL). Note <code>last_spatial_col_index</code> does not need to be specified when creating a PointRDD because it will automatically have the implied value of ( <code>first_spatial_col_index + 1</code> ). For all other types of RDDs, if <code>last_spatial_col_index</code> is unspecified, then it will assume the value of -1 (i.e., the last of all input columns). |
| <code>has_non_spatial_attrs</code>   | Whether the input contains non-spatial attributes.                                                                                                                                                                                                                                                                                                                                                                                                            |
| <code>storage_level</code>           | Storage level of the RDD (default: <code>MEMORY_ONLY</code> ).                                                                                                                                                                                                                                                                                                                                                                                                |
| <code>repartition</code>             | The minimum number of partitions to have in the resulting RDD (default: 1).                                                                                                                                                                                                                                                                                                                                                                                   |

### Value

A typed SpatialRDD.

### See Also

Other Sedona RDD data interface functions: [sedona\\_read\\_geojson\(\)](#), [sedona\\_read\\_shapefile\\_to\\_typed\\_rdd\(\)](#), [sedona\\_save\\_spatial\\_rdd\(\)](#), [sedona\\_write\\_wkb\(\)](#)

### Examples

```

library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

```

```

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your csv file
  rdd <- sedona_read_dsv_to_typed_rdd(
    sc,
    location = input_location,
    delimiter = ",",
    type = "point",
    first_spatial_col_index = 1L
  )
}

```

---

|                     |                                                |
|---------------------|------------------------------------------------|
| sedona_read_geojson | <i>Read geospatial data into a Spatial RDD</i> |
|---------------------|------------------------------------------------|

---

## Description

Import spatial object from an external data source into a Sedona SpatialRDD.

- sedona\_read\_shapefile: from a shapefile
- sedona\_read\_geojson: from a geojson file
- sedona\_read\_wkt: from a geojson file
- sedona\_read\_wkb: from a geojson file

## Usage

```

sedona_read_geojson(
  sc,
  location,
  allow_invalid_geometries = TRUE,
  skip_syntactically_invalid_geometries = TRUE,
  storage_level = "MEMORY_ONLY",
  repartition = 1L
)

```

```

sedona_read_wkb(
  sc,
  location,
  wkb_col_idx = 0L,
  allow_invalid_geometries = TRUE,
  skip_syntactically_invalid_geometries = TRUE,
  storage_level = "MEMORY_ONLY",
  repartition = 1L
)

```

```

sedona_read_wkt(

```

```

    sc,
    location,
    wkt_col_idx = 0L,
    allow_invalid_geometries = TRUE,
    skip_syntactically_invalid_geometries = TRUE,
    storage_level = "MEMORY_ONLY",
    repartition = 1L
  )

  sedona_read_shapefile(sc, location, storage_level = "MEMORY_ONLY")

```

### Arguments

|                                                    |                                                                                                           |
|----------------------------------------------------|-----------------------------------------------------------------------------------------------------------|
| <code>sc</code>                                    | A <code>spark_connection</code> .                                                                         |
| <code>location</code>                              | Location of the data source.                                                                              |
| <code>allow_invalid_geometries</code>              | Whether to allow topology-invalid geometries to exist in the resulting RDD.                               |
| <code>skip_syntactically_invalid_geometries</code> | Whether to allows Sedona to automatically skip syntax-invalid geometries, rather than throwing errorings. |
| <code>storage_level</code>                         | Storage level of the RDD (default: <code>MEMORY_ONLY</code> ).                                            |
| <code>repartition</code>                           | The minimum number of partitions to have in the resulting RDD (default: 1).                               |
| <code>wkb_col_idx</code>                           | Zero-based index of column containing hex-encoded WKB data (default: 0).                                  |
| <code>wkt_col_idx</code>                           | Zero-based index of column containing hex-encoded WKB data (default: 0).                                  |

### Value

A `SpatialRDD`.

### See Also

Other Sedona RDD data interface functions: [sedona\\_read\\_dsv\\_to\\_typed\\_rdd\(\)](#), [sedona\\_read\\_shapefile\\_to\\_typed\\_rdd\(\)](#), [sedona\\_save\\_spatial\\_rdd\(\)](#), [sedona\\_write\\_wkb\(\)](#)

### Examples

```

library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_geojson(sc, location = input_location)
}

```

---

sedona\_read\_shapefile\_to\_typed\_rdd

*(Deprecated) Create a typed SpatialRDD from a shapefile or geojson data source.*

---

## Description

### [Deprecated]

Constructors of typed RDD (PointRDD, PolygonRDD, LineStringRDD) are soft deprecated, use non-types versions

Create a typed SpatialRDD (namely, a PointRDD, a PolygonRDD, or a LineStringRDD)

- sedona\_read\_shapefile\_to\_typed\_rdd: from a shapefile data source
- sedona\_read\_geojson\_to\_typed\_rdd: from a GeoJSON data source

## Usage

```
sedona_read_shapefile_to_typed_rdd(
  sc,
  location,
  type = c("point", "polygon", "linestring"),
  storage_level = "MEMORY_ONLY"
)
```

```
sedona_read_geojson_to_typed_rdd(
  sc,
  location,
  type = c("point", "polygon", "linestring"),
  has_non_spatial_attrs = TRUE,
  storage_level = "MEMORY_ONLY",
  repartition = 1L
)
```

## Arguments

|                       |                                                                              |
|-----------------------|------------------------------------------------------------------------------|
| sc                    | A spark_connection.                                                          |
| location              | Location of the data source.                                                 |
| type                  | Type of the SpatialRDD (must be one of "point", "polygon", or "linestring"). |
| storage_level         | Storage level of the RDD (default: MEMORY_ONLY).                             |
| has_non_spatial_attrs | Whether the input contains non-spatial attributes.                           |
| repartition           | The minimum number of partitions to have in the resulting RDD (default: 1).  |

## Value

A typed SpatialRDD.



**See Also**

Other Sedona RDD data interface functions: [sedona\\_read\\_dsv\\_to\\_typed\\_rdd\(\)](#), [sedona\\_read\\_geojson\(\)](#), [sedona\\_save\\_spatial\\_rdd\(\)](#), [sedona\\_write\\_wkb\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your shapefile
  rdd <- sedona_read_shapefile_to_typed_rdd(
    sc,
    location = input_location, type = "polygon"
  )
}
```

---

sedona\_render\_choropleth\_map

*Visualize a Sedona spatial RDD using a choropleth map.*

---

**Description**

Generate a choropleth map of a pair RDD assigning integral values to polygons.

**Usage**

```
sedona_render_choropleth_map(
  pair_rdd,
  resolution_x,
  resolution_y,
  output_location,
  output_format = c("png", "gif", "svg"),
  boundary = NULL,
  color_of_variation = c("red", "green", "blue"),
  base_color = c(0, 0, 0),
  shade = TRUE,
  reverse_coords = FALSE,
  overlay = NULL,
  browse = interactive()
)
```

**Arguments**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| pair_rdd           | A pair RDD with Sedona Polygon objects being keys and java.lang.Long being values.                                                                                                                                                                                                                                                                                                                          |
| resolution_x       | Resolution on the x-axis.                                                                                                                                                                                                                                                                                                                                                                                   |
| resolution_y       | Resolution on the y-axis.                                                                                                                                                                                                                                                                                                                                                                                   |
| output_location    | Location of the output image. This should be the desired path of the image file excluding extension in its file name.                                                                                                                                                                                                                                                                                       |
| output_format      | File format of the output image. Currently "png", "gif", and "svg" formats are supported (default: "png").                                                                                                                                                                                                                                                                                                  |
| boundary           | Only render data within the given rectangular boundary. The boundary parameter can be set to either a numeric vector of c(min_x, max_x, min_y, max_y) values, or with a bounding box object e.g., new_bounding_box(sc, min_x, max_x, min_y, max_y), or NULL (the default). If boundary is NULL, then the minimum bounding box of the input spatial RDD will be computed and used as boundary for rendering. |
| color_of_variation | Which color channel will vary depending on values of data points. Must be one of "red", "green", or "blue". Default: red.                                                                                                                                                                                                                                                                                   |
| base_color         | Color of any data point with value 0. Must be a numeric vector of length 3 specifying values for red, green, and blue channels. Default: c(0, 0, 0).                                                                                                                                                                                                                                                        |
| shade              | Whether data point with larger magnitude will be displayed with darker color. Default: TRUE.                                                                                                                                                                                                                                                                                                                |
| reverse_coords     | Whether to reverse spatial coordinates in the plot (default: FALSE).                                                                                                                                                                                                                                                                                                                                        |
| overlay            | A viz_op object containing a raster image to be displayed on top of the resulting image.                                                                                                                                                                                                                                                                                                                    |
| browse             | Whether to open the rendered image in a browser (default: interactive()).                                                                                                                                                                                                                                                                                                                                   |

**Value**

No return value.

**See Also**

Other Sedona visualization routines: [sedona\\_render\\_heatmap\(\)](#), [sedona\\_render\\_scatter\\_plot\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  pt_input_location <- "/dev/null" # replace it with the path to your input file
  pt_rdd <- sedona_read_dsv_to_typed_rdd(
```

```

    sc,
    location = pt_input_location,
    type = "point",
    first_spatial_col_index = 1
  )
  polygon_input_location <- "/dev/null" # replace it with the path to your input file
  polygon_rdd <- sedona_read_geojson_to_typed_rdd(
    sc,
    location = polygon_input_location,
    type = "polygon"
  )
  join_result_rdd <- sedona_spatial_join_count_by_key(
    pt_rdd,
    polygon_rdd,
    join_type = "intersect",
    partitioner = "quadtree"
  )
  sedona_render_choropleth_map(
    join_result_rdd,
    400,
    200,
    output_location = tempfile("choropleth-map-"),
    boundary = c(-86.8, -86.6, 33.4, 33.6),
    base_color = c(255, 255, 255)
  )
}

```

---

sedona\_render\_heatmap *Visualize a Sedona spatial RDD using a heatmap.*

---

## Description

Generate a heatmap of geometrical object(s) within a Sedona spatial RDD.

## Usage

```

sedona_render_heatmap(
  rdd,
  resolution_x,
  resolution_y,
  output_location,
  output_format = c("png", "gif", "svg"),
  boundary = NULL,
  blur_radius = 10L,
  overlay = NULL,
  browse = interactive()
)

```

**Arguments**

|                 |                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rdd             | A Sedona spatial RDD.                                                                                                                                                                                                                                                                                                                                                                                       |
| resolution_x    | Resolution on the x-axis.                                                                                                                                                                                                                                                                                                                                                                                   |
| resolution_y    | Resolution on the y-axis.                                                                                                                                                                                                                                                                                                                                                                                   |
| output_location | Location of the output image. This should be the desired path of the image file excluding extension in its file name.                                                                                                                                                                                                                                                                                       |
| output_format   | File format of the output image. Currently "png", "gif", and "svg" formats are supported (default: "png").                                                                                                                                                                                                                                                                                                  |
| boundary        | Only render data within the given rectangular boundary. The boundary parameter can be set to either a numeric vector of c(min_x, max_x, min_y, max_y) values, or with a bounding box object e.g., new_bounding_box(sc, min_x, max_x, min_y, max_y), or NULL (the default). If boundary is NULL, then the minimum bounding box of the input spatial RDD will be computed and used as boundary for rendering. |
| blur_radius     | Controls the radius of a Gaussian blur in the resulting heatmap.                                                                                                                                                                                                                                                                                                                                            |
| overlay         | A viz_op object containing a raster image to be displayed on top of the resulting image.                                                                                                                                                                                                                                                                                                                    |
| browse          | Whether to open the rendered image in a browser (default: interactive()).                                                                                                                                                                                                                                                                                                                                   |

**Value**

No return value.

**See Also**

Other Sedona visualization routines: [sedona\\_render\\_choropleth\\_map\(\)](#), [sedona\\_render\\_scatter\\_plot\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_dsv_to_typed_rdd(
    sc,
    location = input_location,
    type = "point"
  )

  sedona_render_heatmap(
    rdd,
    resolution_x = 800,
    resolution_y = 600,
    output_location = tempfile("points-"),

```

```

        output_format = "png",
        boundary = c(-91, -84, 30, 35),
        blur_radius = 10
    )
}

```

---

sedona\_render\_scatter\_plot

*Visualize a Sedona spatial RDD using a scatter plot.*

---

## Description

Generate a scatter plot of geometrical object(s) within a Sedona spatial RDD.

## Usage

```

sedona_render_scatter_plot(
  rdd,
  resolution_x,
  resolution_y,
  output_location,
  output_format = c("png", "gif", "svg"),
  boundary = NULL,
  color_of_variation = c("red", "green", "blue"),
  base_color = c(0, 0, 0),
  shade = TRUE,
  reverse_coords = FALSE,
  overlay = NULL,
  browse = interactive()
)

```

## Arguments

|                 |                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| rdd             | A Sedona spatial RDD.                                                                                                                                                                                                                                                                                                                                                                                       |
| resolution_x    | Resolution on the x-axis.                                                                                                                                                                                                                                                                                                                                                                                   |
| resolution_y    | Resolution on the y-axis.                                                                                                                                                                                                                                                                                                                                                                                   |
| output_location | Location of the output image. This should be the desired path of the image file excluding extension in its file name.                                                                                                                                                                                                                                                                                       |
| output_format   | File format of the output image. Currently "png", "gif", and "svg" formats are supported (default: "png").                                                                                                                                                                                                                                                                                                  |
| boundary        | Only render data within the given rectangular boundary. The boundary parameter can be set to either a numeric vector of c(min_x, max_y, min_y, max_y) values, or with a bounding box object e.g., new_bounding_box(sc, min_x, max_y, min_y, max_y), or NULL (the default). If boundary is NULL, then the minimum bounding box of the input spatial RDD will be computed and used as boundary for rendering. |

|                    |                                                                                                                                                      |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| color_of_variation | Which color channel will vary depending on values of data points. Must be one of "red", "green", or "blue". Default: red.                            |
| base_color         | Color of any data point with value 0. Must be a numeric vector of length 3 specifying values for red, green, and blue channels. Default: c(0, 0, 0). |
| shade              | Whether data point with larger magnitude will be displayed with darker color. Default: TRUE.                                                         |
| reverse_coords     | Whether to reverse spatial coordinates in the plot (default: FALSE).                                                                                 |
| overlay            | A viz_op object containing a raster image to be displayed on top of the resulting image.                                                             |
| browse             | Whether to open the rendered image in a browser (default: interactive()).                                                                            |

**Value**

No return value.

**See Also**

Other Sedona visualization routines: [sedona\\_render\\_choropleth\\_map\(\)](#), [sedona\\_render\\_heatmap\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_dsv_to_typed_rdd(
    sc,
    location = input_location,
    type = "point"
  )

  sedona_render_scatter_plot(
    rdd,
    resolution_x = 800,
    resolution_y = 600,
    output_location = tempfile("points-"),
    output_format = "png",
    boundary = c(-91, -84, 30, 35)
  )
}
```

---

sedona\_save\_spatial\_rdd

*Save a Spark dataframe containing exactly 1 spatial column into a file.*


---

## Description

Export serialized data from a Spark dataframe containing exactly 1 spatial column into a file.

## Usage

```
sedona_save_spatial_rdd(
  x,
  spatial_col,
  output_location,
  output_format = c("wkb", "wkt", "geojson")
)
```

## Arguments

|                 |                                                                                            |
|-----------------|--------------------------------------------------------------------------------------------|
| x               | A Spark dataframe object in sparklyr or a dplyr expression representing a Spark SQL query. |
| spatial_col     | The name of the spatial column.                                                            |
| output_location | Location of the output file.                                                               |
| output_format   | Format of the output.                                                                      |

## Value

No return value.

## See Also

Other Sedona RDD data interface functions: [sedona\\_read\\_dsv\\_to\\_typed\\_rdd\(\)](#), [sedona\\_read\\_geojson\(\)](#), [sedona\\_read\\_shapefile\\_to\\_typed\\_rdd\(\)](#), [sedona\\_write\\_wkb\(\)](#)

## Examples

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  tbl <- dplyr::tbl(
    sc,
    dplyr::sql("SELECT ST_GeomFromText('POINT(-71.064544 42.28787)') AS `pt`")
  )
  sedona_save_spatial_rdd(
```

```
tbl %>% dplyr::mutate(id = 1),
  spatial_col = "pt",
  output_location = "/tmp/pts.wkb",
  output_format = "wkb"
)
}
```

---

|                     |                                                                     |
|---------------------|---------------------------------------------------------------------|
| sedona_spatial_join | <i>Perform a spatial join operation on two Sedona spatial RDDs.</i> |
|---------------------|---------------------------------------------------------------------|

---

### Description

Given `spatial_rdd` and `query_window_rdd`, return a pair RDD containing all pairs of geometrical elements (p, q) such that p is an element of `spatial_rdd`, q is an element of `query_window_rdd`, and (p, q) satisfies the spatial relation specified by `join_type`.

### Usage

```
sedona_spatial_join(
  spatial_rdd,
  query_window_rdd,
  join_type = c("contain", "intersect"),
  partitioner = c("quadtree", "kdbtree"),
  index_type = c("quadtree", "rtree")
)
```

### Arguments

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>spatial_rdd</code>      | Spatial RDD containing geometries to be queried.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <code>query_window_rdd</code> | Spatial RDD containing the query window(s).                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <code>join_type</code>        | Type of the join query (must be either "contain" or "intersect"). If <code>join_type</code> is "contain", then a geometry from <code>spatial_rdd</code> will match a geometry from the <code>query_window_rdd</code> if and only if the former is fully contained in the latter. If <code>join_type</code> is "intersect", then a geometry from <code>spatial_rdd</code> will match a geometry from the <code>query_window_rdd</code> if and only if the former intersects the latter. |
| <code>partitioner</code>      | Spatial partitioning to apply to both <code>spatial_rdd</code> and <code>query_window_rdd</code> to facilitate the join query. Can be either a grid type (currently "quadtree" and "kdbtree" are supported) or a custom spatial partitioner object. If <code>partitioner</code> is NULL, then assume the same spatial partitioner has been applied to both <code>spatial_rdd</code> and <code>query_window_rdd</code> already and skip the partitioning step.                          |
| <code>index_type</code>       | Controls how <code>spatial_rdd</code> and <code>query_window_rdd</code> will be indexed (unless they are indexed already). If "NONE", then no index will be constructed and matching geometries will be identified in a doubly nested- loop iterating through all possible pairs of elements from <code>spatial_rdd</code> and <code>query_window_rdd</code> , which will be inefficient for large data sets.                                                                          |



**Value**

A spatial RDD containing the join result.

**See Also**

Other Sedona spatial join operator: [sedona\\_spatial\\_join\\_count\\_by\\_key\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_dsv_to_typed_rdd(
    sc,
    location = input_location,
    delimiter = ",",
    type = "point",
    first_spatial_col_index = 1L
  )
  query_rdd_input_location <- "/dev/null" # replace it with the path to your input file
  query_rdd <- sedona_read_shapefile_to_typed_rdd(
    sc,
    location = query_rdd_input_location,
    type = "polygon"
  )
  join_result_rdd <- sedona_spatial_join(
    rdd,
    query_rdd,
    join_type = "intersect",
    partitioner = "quadtree"
  )
}
```

---

sedona\_spatial\_join\_count\_by\_key

*Perform a spatial count-by-key operation based on two Sedona spatial RDDs.*

---

**Description**

For each element  $p$  from `spatial_rdd`, count the number of unique elements  $q$  from `query_window_rdd` such that  $(p, q)$  satisfies the spatial relation specified by `join_type`.

**Usage**

```
sedona_spatial_join_count_by_key(
  spatial_rdd,
  query_window_rdd,
  join_type = c("contain", "intersect"),
  partitioner = c("quadtree", "kdmtree"),
  index_type = c("quadtree", "rtree")
)
```

**Arguments**

|                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>spatial_rdd</code>      | Spatial RDD containing geometries to be queried.                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <code>query_window_rdd</code> | Spatial RDD containing the query window(s).                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <code>join_type</code>        | Type of the join query (must be either "contain" or "intersect"). If <code>join_type</code> is "contain", then a geometry from <code>spatial_rdd</code> will match a geometry from the <code>query_window_rdd</code> if and only if the former is fully contained in the latter. If <code>join_type</code> is "intersect", then a geometry from <code>spatial_rdd</code> will match a geometry from the <code>query_window_rdd</code> if and only if the former intersects the latter. |
| <code>partitioner</code>      | Spatial partitioning to apply to both <code>spatial_rdd</code> and <code>query_window_rdd</code> to facilitate the join query. Can be either a grid type (currently "quadtree" and "kdmtree" are supported) or a custom spatial partitioner object. If <code>partitioner</code> is NULL, then assume the same spatial partitioner has been applied to both <code>spatial_rdd</code> and <code>query_window_rdd</code> already and skip the partitioning step.                          |
| <code>index_type</code>       | Controls how <code>spatial_rdd</code> and <code>query_window_rdd</code> will be indexed (unless they are indexed already). If "NONE", then no index will be constructed and matching geometries will be identified in a doubly nested- loop iterating through all possible pairs of elements from <code>spatial_rdd</code> and <code>query_window_rdd</code> , which will be inefficient for large data sets.                                                                          |

**Value**

A spatial RDD containing the join-count-by-key results.

**See Also**

Other Sedona spatial join operator: [sedona\\_spatial\\_join\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
```

```

rdd <- sedona_read_dsv_to_typed_rdd(
  sc,
  location = input_location,
  delimiter = ",",
  type = "point",
  first_spatial_col_index = 1L
)
query_rdd_input_location <- "/dev/null" # replace it with the path to your input file
query_rdd <- sedona_read_shapefile_to_typed_rdd(
  sc,
  location = query_rdd_input_location,
  type = "polygon"
)
join_result_rdd <- sedona_spatial_join_count_by_key(
  rdd,
  query_rdd,
  join_type = "intersect",
  partitioner = "quadtree"
)
}

```

---

sedona\_write\_wkb

*Write SpatialRDD into a file.*


---

## Description

Export serialized data from a Sedona SpatialRDD into a file.

- sedona\_write\_wkb:
- sedona\_write\_wkt:
- sedona\_write\_geojson:

## Usage

```
sedona_write_wkb(x, output_location)
```

```
sedona_write_wkt(x, output_location)
```

```
sedona_write_geojson(x, output_location)
```

## Arguments

|                 |                              |
|-----------------|------------------------------|
| x               | The SpatialRDD object.       |
| output_location | Location of the output file. |

## Value

No return value.

**See Also**

Other Sedona RDD data interface functions: [sedona\\_read\\_dsv\\_to\\_typed\\_rdd\(\)](#), [sedona\\_read\\_geojson\(\)](#), [sedona\\_read\\_shapefile\\_to\\_typed\\_rdd\(\)](#), [sedona\\_save\\_spatial\\_rdd\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- sedona_read_wkb(
    sc,
    location = input_location,
    wkb_col_idx = 0L
  )
  sedona_write_wkb(rdd, "/tmp/wkb_output.tsv")
}
```

---

spark\_read\_shapefile    *Read geospatial data into a Spark DataFrame.*

---

**Description****[Deprecated]**

These functions are deprecated and will be removed in a future release. Sedona has been implementing readers as spark DataFrame sources, so you can use `spark_read_source` with the right sources ("shapefile", "geojson", "geoparquet") to read geospatial data.

Functions to read geospatial data from a variety of formats into Spark DataFrames.

- `spark_read_shapefile`: from a shapefile
- `spark_read_geojson`: from a geojson file
- `spark_read_geoparquet`: from a geoparquet file

**Usage**

```
spark_read_shapefile(sc, name = NULL, path = name, options = list(), ...)
```

```
spark_read_geojson(
  sc,
  name = NULL,
  path = name,
  options = list(),
  repartition = 0,
```

```

    memory = TRUE,
    overwrite = TRUE
  )

  spark_read_geoparquet(
    sc,
    name = NULL,
    path = name,
    options = list(),
    repartition = 0,
    memory = TRUE,
    overwrite = TRUE
  )

```

### Arguments

|             |                                                                                                                                                                                                                            |
|-------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| sc          | A spark_connection.                                                                                                                                                                                                        |
| name        | The name to assign to the newly generated table.                                                                                                                                                                           |
| path        | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.                                                                                                   |
| options     | A list of strings with additional options. See <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration">https://spark.apache.org/docs/latest/sql-programming-guide.html#configuration</a> . |
| ...         | Optional arguments; currently unused.                                                                                                                                                                                      |
| repartition | The number of partitions used to distribute the generated table. Use 0 (the default) to avoid partitioning.                                                                                                                |
| memory      | Boolean; should the data be loaded eagerly into memory? (That is, should the table be cached?)                                                                                                                             |
| overwrite   | Boolean; overwrite the table with the given name if it already exists?                                                                                                                                                     |

### Value

A tbl

### See Also

Other Sedona DF data interface functions: [spark\\_write\\_geojson\(\)](#)

### Examples

```

library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  input_location <- "/dev/null" # replace it with the path to your input file
  rdd <- spark_read_shapefile(sc, location = input_location)
}

```

---

|                     |                                                      |
|---------------------|------------------------------------------------------|
| spark_write_geojson | <i>Write geospatial data from a Spark DataFrame.</i> |
|---------------------|------------------------------------------------------|

---

## Description

### [Deprecated]

These functions are deprecated and will be removed in a future release. Sedona has been implementing writers as spark DataFrame sources, so you can use `spark_write_source` with the right sources ("shapefile", "geojson", "geoparquet") to write geospatial data.

Functions to write geospatial data into a variety of formats from Spark DataFrames.

- `spark_write_geojson`: to GeoJSON
- `spark_write_geoparquet`: to GeoParquet
- `spark_write_raster`: to raster tiles after using RS output functions (RS\_AsXXX)

## Usage

```
spark_write_geojson(  
  x,  
  path,  
  mode = NULL,  
  options = list(),  
  partition_by = NULL,  
  ...  
)  
  
spark_write_geoparquet(  
  x,  
  path,  
  mode = NULL,  
  options = list(),  
  partition_by = NULL,  
  ...  
)  
  
spark_write_raster(  
  x,  
  path,  
  mode = NULL,  
  options = list(),  
  partition_by = NULL,  
  ...  
)
```

**Arguments**

|              |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| x            | A Spark DataFrame or dplyr operation                                                                                                                                                                                                                                                                                                                                                                                                           |
| path         | The path to the file. Needs to be accessible from the cluster. Supports the "hdfs://", "s3a://" and "file://" protocols.                                                                                                                                                                                                                                                                                                                       |
| mode         | A character element. Specifies the behavior when data or table already exists. Supported values include: 'error', 'append', 'overwrite' and ignore. Notice that 'overwrite' will also change the column structure.<br>For more details see also <a href="https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes">https://spark.apache.org/docs/latest/sql-programming-guide.html#save-modes</a> for your version of Spark. |
| options      | A list of strings with additional options.                                                                                                                                                                                                                                                                                                                                                                                                     |
| partition_by | A character vector. Partitions the output by the given columns on the file system.                                                                                                                                                                                                                                                                                                                                                             |
| ...          | Optional arguments; currently unused.                                                                                                                                                                                                                                                                                                                                                                                                          |

**See Also**

Other Sedona DF data interface functions: [spark\\_read\\_shapefile\(\)](#)

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  tbl <- dplyr::tbl(
    sc,
    dplyr::sql("SELECT ST_GeomFromText('POINT(-71.064544 42.28787)') AS `pt`")
  )
  spark_write_geojson(
    tbl %>% dplyr::mutate(id = 1),
    output_location = "/tmp/pts.geojson"
  )
}
```

---

|                |                                                                                  |
|----------------|----------------------------------------------------------------------------------|
| to_spatial_rdd | <i>Export a Spark SQL query with a spatial column into a Sedona spatial RDD.</i> |
|----------------|----------------------------------------------------------------------------------|

---

**Description**

Given a Spark dataframe object or a dplyr expression encapsulating a Spark SQL query, build a Sedona spatial RDD that will encapsulate the same query or data source. The input should contain exactly one spatial column and all other non-spatial columns will be treated as custom user-defined attributes in the resulting spatial RDD.

**Usage**

```
to_spatial_rdd(x, spatial_col)
```

**Arguments**

|             |                                                                                            |
|-------------|--------------------------------------------------------------------------------------------|
| x           | A Spark dataframe object in sparklyr or a dplyr expression representing a Spark SQL query. |
| spatial_col | The name of the spatial column.                                                            |

**Value**

A SpatialRDD encapsulating the query.

**Examples**

```
library(sparklyr)
library(apache.sedona)

sc <- spark_connect(master = "spark://HOST:PORT")

if (!inherits(sc, "test_connection")) {
  tbl <- dplyr::tbl(
    sc,
    dplyr::sql("SELECT ST_GeomFromText('POINT(-71.064544 42.28787)') AS `pt`")
  )
  rdd <- to_spatial_rdd(tbl, "pt")
}
```



# Index

- \* **Sedona DF data interface functions**
  - spark\_read\_shapefile, [28](#)
  - spark\_write\_geojson, [30](#)
- \* **Sedona RDD data interface functions**
  - sedona\_read\_dsv\_to\_typed\_rdd, [12](#)
  - sedona\_read\_geojson, [14](#)
  - sedona\_read\_shapefile\_to\_typed\_rdd, [16](#)
  - sedona\_save\_spatial\_rdd, [23](#)
  - sedona\_write\_wkb, [27](#)
- \* **Sedona spatial join operator**
  - sedona\_spatial\_join, [24](#)
  - sedona\_spatial\_join\_count\_by\_key, [25](#)
- \* **Sedona spatial query**
  - sedona\_knn\_query, [9](#)
  - sedona\_range\_query, [11](#)
- \* **Sedona visualization routines**
  - sedona\_render\_choropleth\_map, [17](#)
  - sedona\_render\_heatmap, [19](#)
  - sedona\_render\_scatter\_plot, [21](#)
- \* **Spatial RDD aggregation routine**
  - approx\_count, [2](#)
  - minimum\_bounding\_box, [4](#)
- approx\_count, [2, 4](#)
- as.spark.dataframe  
(sdf\_register.spatial\_rdd), [6](#)
- crs\_transform, [3](#)
- minimum\_bounding\_box, [2, 4](#)
- new\_bounding\_box, [5](#)
- sdf\_register.spatial\_rdd, [6](#)
- sedona\_apply\_spatial\_partitioner, [7](#)
- sedona\_build\_index, [8](#)
- sedona\_knn\_query, [9, 11](#)
- sedona\_range\_query, [10, 11](#)
- sedona\_read\_dsv\_to\_typed\_rdd, [12, 15, 17, 23, 28](#)
- sedona\_read\_geojson, [13, 14, 17, 23, 28](#)
- sedona\_read\_geojson\_to\_typed\_rdd  
(sedona\_read\_shapefile\_to\_typed\_rdd), [16](#)
- sedona\_read\_shapefile  
(sedona\_read\_geojson), [14](#)
- sedona\_read\_shapefile\_to\_typed\_rdd, [13, 15, 16, 23, 28](#)
- sedona\_read\_wkb (sedona\_read\_geojson), [14](#)
- sedona\_read\_wkt (sedona\_read\_geojson), [14](#)
- sedona\_render\_choropleth\_map, [17, 20, 22](#)
- sedona\_render\_heatmap, [18, 19, 22](#)
- sedona\_render\_scatter\_plot, [18, 20, 21](#)
- sedona\_save\_spatial\_rdd, [13, 15, 17, 23, 28](#)
- sedona\_spatial\_join, [24, 26](#)
- sedona\_spatial\_join\_count\_by\_key, [25, 25](#)
- sedona\_write\_geojson  
(sedona\_write\_wkb), [27](#)
- sedona\_write\_wkb, [13, 15, 17, 23, 27](#)
- sedona\_write\_wkt (sedona\_write\_wkb), [27](#)
- spark\_read\_geojson  
(spark\_read\_shapefile), [28](#)
- spark\_read\_geoparquet  
(spark\_read\_shapefile), [28](#)
- spark\_read\_shapefile, [28, 31](#)
- spark\_write\_geojson, [29, 30](#)
- spark\_write\_geoparquet  
(spark\_write\_geojson), [30](#)
- spark\_write\_raster  
(spark\_write\_geojson), [30](#)
- to\_spatial\_rdd, [31](#)