Package 'duckdbfs'

May 8, 2026

Title High Performance Remote File System, Database and 'Geospatial' Access Using 'duckdb'

Version 0.1.2

Description Provides friendly wrappers for creating 'duckdb'-backed connections to tabular datasets ('csv', parquet, etc) on local or remote file systems. This mimics the behaviour of ``open_dataset'' in the 'arrow' package, but in addition to 'S3' file system also generalizes to any list of 'http' URLs.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

URL https://github.com/cboettig/duckdbfs, https://cboettig.github.io/duckdbfs/

BugReports https://github.com/cboettig/duckdbfs/issues

Depends R (>= 4.2)

Imports DBI, dbplyr, dplyr, duckdb (>= 1.1), fs, glue

Suggests curl, sf, jsonlite, spelling, miniocclient, testthat (>= 3.0.0)

Config/testthat/edition 3

Language en-US

NeedsCompilation no

Author Carl Boettiger [aut, cre] (ORCID: <https://orcid.org/0000-0002-1642-628X>), Michael D. Sumner [ctb] (ORCID: <https://orcid.org/0000-0002-2471-7511>)

Maintainer Carl Boettiger <cboettig@gmail.com>

Repository CRAN

Date/Publication 2025-10-12 06:20:02 UTC

Contents

`as_dataset` *as_dataset*

Description

Push a local (in-memory) dataset into a the duckdb database as a table. This enables it to share the connection source with other data. This is equivalent to the behavior of copy=TRUE on many (but not all) of the two-table verbs in dplyr.

Usage

```
as_dataset(df, conn = cached_connection())
```

Arguments

`df`	a local data frame. Otherwise will be passed back without side effects
`conn`	A connection to a database.

Value

a remote `dplyr::tbl` connection to the table.

as_view *as_view*

Description

Create a View of the current query. This can be an effective way to allow a query chain to remain lazy

Usage

```
as_view(x, tblname = tmp_tbl_name(), conn = cached_connection())
```

Arguments

x	a duckdb spatial dataset
tblname	The name of the table to create in the database.
conn	A connection to a database.

Examples

```
path <- system.file("extdata/spatial-test.csv", package="duckdbfs")
df <- open_dataset(path)
library(dplyr)

df |> filter(latitude > 5) |> as_view()
```

cached_connection *create a cachable duckdb connection*

Description

This function is primarily intended for internal use by other duckdbfs functions. However, it can be called directly by the user whenever it is desirable to have direct access to the connection object.

Usage

```
cached_connection(
  dbdir = ":memory:",
  read_only = FALSE,
  bigint = "numeric",
  config = list(temp_directory = tempfile()),
  autoload_exts = getOption("duckdbfs_autoload_extensions", TRUE),
 with_spatial = not_windows() && getOption("duckdbfs_autoload_extensions", TRUE),
  with_h3 = not_windows() && getOption("duckdbfs_autoload_extensions", TRUE)
)
```

Arguments

dbdir	Location for database files. Should be a path to an existing directory in the file system. With the default (or ""), all data is kept in RAM.
read_only	Set to TRUE for read-only operation. For file-based databases, this is only applied when the database file is opened for the first time. Subsequent connections (via the same drv object or a drv object pointing to the same path) will silently ignore this flag.
bigint	How 64-bit integers should be returned. There are two options: "numeric" and "integer64". If "numeric" is selected, bigint integers will be treated as double/numeric. If "integer64" is selected, bigint integers will be set to bit64 encoding.
config	Named list with DuckDB configuration flags, see https://duckdb.org/docs/configuration/overview#configuration-reference for the possible options. These flags are only applied when the database object is instantiated. Subsequent connections will silently ignore these flags.
autoload_exts	should we auto-load extensions? TRUE by default, can be configured with options(duckdbfs_autoload_extensions = FALSE)
with_spatial	install (if missing) and load spatial extension, default TRUE Opt out by closing any active cached connection first (with close_connection()) and re-instantiating the with connect(with_spatial = FALSE).
with_h3	install (if missing) and load the h3 spatial index extension. Default TRUE

Details

When first called (by a user or internal function), this function both creates a duckdb connection and places that connection into a cache (duckdbfs_conn option). On subsequent calls, this function returns the cached connection, rather than recreating a fresh connection.

This frees the user from the responsibility of managing a connection object, because functions needing access to the connection can use this to create or access the existing connection. At the close of the global environment, this function's finalizer should gracefully shutdown the connection before removing the cache.

By default, this function creates an in-memory connection. When reading from on-disk or remote files (parquet or csv), this option can still effectively support most operations on much-larger-than-RAM data. However, some operations require additional working space, so by default we set a temporary storage location in configuration as well.

Value

a duckdb::duckdb() connection object

Examples

```
con <- cached_connection()
close_connection(con)
```

close_connection *close connection*

Description

close connection

Usage

```
close_connection(conn = cached_connection())
```

Arguments

conn	a duckdb connection (leave blank) Closes the invisible cached connection to duckdb

Details

Shuts down connection before gc removes it. Then clear cached reference to avoid using a stale connection This avoids complaint about connection being garbage collected.

Value

returns nothing.

Examples

```
close_connection()
```

duckdb_config *duckdb configuration*

Description

duckdb configuration

Usage

```
duckdb_config(..., conn = cached_connection())
```

Arguments

...	named argument of the parameters to set, see examples see all possible configuration options at https://duckdb.org/docs/sql/configuration.html
conn	A connection to a database.

Details

Note: in I/O bound tasks such as streaming data, it can be helpful to set thread parallelism significantly higher than available CPU cores.

Value

the active duckdb connection, invisibly

See Also

duckdb_reset, duckdb_get_config

Examples

```
duckdb_config(threads = 1, memory_limit = '10GB')
duckdb_get_config("threads")
duckdb_reset("threads")
```

---

duckdb_extensions *show duckdb extensions*

Description

show duckdb extensions

Usage

```
duckdb_extensions(conn = cached_connection())
```

Arguments

conn	A connection to a database.

Value

a data frame listing all available extensions, with boolean columns indicating which extensions are installed or loaded, and a description of each extension.

Examples

```
duckdb_extensions()
```

duckdb_get_config *duckdb reset configuration to default*

Description

duckdb reset configuration to default

Usage

```
duckdb_get_config(x = NULL, conn = cached_connection())
```

Arguments

x	parameter name. Omit to see a table of all settings.
conn	A connection to a database.

See Also

duckdb_config, duckdb_get_config

Examples

```
# Full config table
duckdb_get_config()

# look up single config value
duckdb_get_config("threads")

# set a different value, test, reset.
duckdb_config(threads = 10)
duckdb_get_config("threads")
duckdb_reset("threads")
```

duckdb_reset *duckdb reset configuration to default*

Description

duckdb reset configuration to default

Usage

```
duckdb_reset(x, conn = cached_connection())
```

Arguments

x	parameter name
conn	A connection to a database.

See Also

duckdb_config, duckdb_get_config

Examples

```
duckdb_config(threads = 10)
duckdb_get_config("threads")
duckdb_reset("threads")
```

---

duckdb_s3_config *Configure S3 settings for database connection*

Description

This function is used to configure S3 settings for a database connection. It allows you to set various S3-related parameters such as access key, secret access key, endpoint, region, session token, uploader settings, URL compatibility mode, URL style, and SSL usage.

Usage

```
duckdb_s3_config(
  conn = cached_connection(),
  s3_access_key_id = NULL,
  s3_secret_access_key = NULL,
  s3_endpoint = NULL,
  s3_region = NULL,
  s3_session_token = NULL,
  s3_uploader_max_filesize = NULL,
  s3_uploader_max_parts_per_file = NULL,
  s3_uploader_thread_limit = NULL,
  s3_url_compatibility_mode = NULL,
  s3_url_style = NULL,
  s3_use_ssl = NULL,
  anonymous = NULL
)
```

Arguments

conn	A database connection object created using the cache_connection function (default: cache_connection()).
s3_access_key_id	The S3 access key ID (default: NULL).
s3_secret_access_key	The S3 secret access key (default: NULL).
s3_endpoint	The S3 endpoint (default: NULL).
s3_region	The S3 region (default: NULL).
s3_session_token	The S3 session token (default: NULL).
s3_uploader_max_filesize	The maximum filesize for S3 uploader (between 50GB and 5TB, default 800GB).
s3_uploader_max_parts_per_file	The maximum number of parts per file for S3 uploader (between 1 and 10000, default 10000).
s3_uploader_thread_limit	The thread limit for S3 uploader (default: 50).
s3_url_compatibility_mode	Disable Globs and Query Parameters on S3 URLs (default: 0, allows globs/queries).
s3_url_style	The style of S3 URLs to use. Default is "vhost" unless s3_endpoint is set, which makes default "path" (i.e. MINIO systems).
s3_use_ssl	Enable or disable SSL for S3 connections (default: 1 (TRUE)).
anonymous	request anonymous access (sets s3_access_key_id and s3_secret_access_key to "", allowing anonymous access to public buckets).

Details

see https://duckdb.org/docs/sql/configuration.html

Value

Returns silently (NULL) if successful.

Examples

```
# Configure S3 settings
duckdb_s3_config(
           s3_access_key_id = "YOUR_ACCESS_KEY_ID",
           s3_secret_access_key = "YOUR_SECRET_ACCESS_KEY",
           s3_endpoint = "YOUR_S3_ENDPOINT",
           s3_region = "YOUR_S3_REGION",
           s3_uploader_max_filesize = "800GB",
           s3_uploader_max_parts_per_file = 100,
           s3_uploader_thread_limit = 8,
           s3_url_compatibility_mode = FALSE,
           s3_url_style = "vhost",
```

```
  s3_use_ssl = TRUE,
  anonymous = TRUE)
```

---

duckdb_secrets *duckdb secrets*

Description

Configure the duckdb secrets for remote access.

Usage

```
duckdb_secrets(
  key = Sys.getenv("AWS_ACCESS_KEY_ID", ""),
  secret = Sys.getenv("AWS_SECRET_ACCESS_KEY", ""),
  endpoint = Sys.getenv("AWS_S3_ENDPOINT", "s3.amazonaws.com"),
  region = Sys.getenv("AWS_REGION", "us-east-1"),
  bucket = NULL,
  url_style = NULL,
  use_ssl = Sys.getenv("AWS_HTTPS", "TRUE"),
  url_compatibility_mode = TRUE,
  session_token = Sys.getenv("AWS_SESSION_TOKEN", ""),
  type = "S3",
  conn = cached_connection()
)
```

Arguments

key	key
secret	secret
endpoint	endpoint address
region	AWS region (ignored by some other S3 providers)
bucket	restricts the "SCOPE" of this key to only objects in this bucket-name. note that the bucket name is currently insensitive to endpoint
url_style	path or vhost, for S3
use_ssl	Use SSL address (https instead of http), default TRUE
url_compatibility_mode	optional mode for increased compatibility with some endpoints
session_token	AWS session token, used in some AWS authentication with short-lived tokens
type	Key type, e.g. S3. See duckdb docs for details. references https://duckdb.org/docs/configuration/secrets_manager.html
conn	A connection to a database.

load_h3 *load the duckdb geospatial data plugin*

Description

load the duckdb geospatial data plugin

Usage

```
load_h3(
  conn = cached_connection(),
  repo = "http://community-extensions.duckdb.org"
)
```

Arguments

conn	A database connection object created using the cache_connection function (default: cache_connection()).
repo	repository path for community extensions

Value

loads the extension and returns status invisibly.

References

https://github.com/isaacbrodsky/h3-duckdb

Examples

```
library(dplyr)
load_h3()
ex <- system.file("extdata/spatial-test.csv", package="duckdbfs")

zoom <- 9L # Zoom must be explicit integer, L
query <- ex |>
  open_dataset(format = "csv") |>
  mutate(h3id = h3_latlng_to_cell_string(latitude, longitude, zoom))

 # as data.frame
 collect(query)

 # write to a file
 path <- tempfile(fileext = ".h3j")
 query |> to_h3j(path)
```

load_spatial *load the duckdb geospatial data plugin*

Description

load the duckdb geospatial data plugin

Usage

```
load_spatial(
  conn = cached_connection(),
  nightly = getOption("duckdbfs_use_nightly", FALSE),
  force = FALSE
)
```

Arguments

conn	A database connection object created using the cache_connection function (default: cache_connection()).
nightly	should we use the nightly version or not? default FALSE, configurable as duckdbfs_use_nightly option.
force	force re-install?

Value

loads the extension and returns status invisibly.

References

https://duckdb.org/docs/extensions/spatial.html

open_dataset *Open a dataset from a variety of sources*

Description

This function opens a dataset from a variety of sources, including Parquet, CSV, etc, using either local file system paths, URLs, or S3 bucket URI notation.

Usage

```
open_dataset(
  sources,
  schema = NULL,
  hive_style = TRUE,
  unify_schemas = FALSE,
  format = c("parquet", "csv", "tsv", "sf"),
  conn = cached_connection(),
  tblname = tmp_tbl_name(),
  mode = "VIEW",
  filename = FALSE,
  recursive = TRUE,
  parser_options = list(),
  ...
)
```

Arguments

sources	A character vector of paths to the dataset files.
schema	The schema for the dataset. If NULL, the schema will be inferred from the dataset files.
hive_style	A logical value indicating whether to the dataset uses Hive-style partitioning.
unify_schemas	A logical value indicating whether to unify the schemas of the dataset files (union_by_name). If TRUE, will execute a UNION by column name across all files (NOTE: this can add considerably to the initial execution time)
format	The format of the dataset files. One of "parquet", "csv", or "sf" (spatial vector files supported by the sf package / GDAL). if no argument is provided, the function will try to guess the type based on minimal heuristics.
conn	A connection to a database.
tblname	The name of the table to create in the database.
mode	The mode to create the table in. One of "VIEW" or "TABLE". Creating a VIEW, the default, will execute more quickly because it does not create a local copy of the dataset. TABLE will create a local copy in duckdb's native format, downloading the full dataset if necessary. When using TABLE mode with large data, please be sure to use a conn connections with disk-based storage, e.g. by calling cached_connection(), e.g. cached_connection("storage_path"), otherwise the full data must fit into RAM. Using TABLE assumes familiarity with R's DBI-based interface.
filename	A logical value indicating whether to include the filename in the table name.
recursive	should we assume recursive path? default TRUE. Set to FALSE if trying to open a single, un-partitioned file.
parser_options	additional options passed to the parser, e.g. to read_csv(), see https://duckdb.org/docs/stable/data/csv/overview.html#parameters
...	optional additional arguments passed to duckdb_s3_config(). Note these apply after those set by the URI notation and thus may be used to override or provide settings not supported in that format.

Value

A lazy dplyr::tbl object representing the opened dataset backed by a duckdb SQL connection. Most dplyr (and some tidyr) verbs can be used directly on this object, as they can be translated into SQL commands automatically via dbplyr. Generic R commands require using dplyr::collect() on the table, which forces evaluation and reading the resulting data into memory.

Examples

```
# A remote, hive-partitioned Parquet dataset
base <- paste0("https://github.com/duckdb/duckdb/raw/main/",
             "data/parquet-testing/hive-partitioning/union_by_name/")
f1 <- paste0(base, "x=1/f1.parquet")
f2 <- paste0(base, "x=1/f2.parquet")
f3 <- paste0(base, "x=2/f2.parquet")

open_dataset(c(f1,f2,f3), unify_schemas = TRUE)

# Access an S3 database specifying an independently-hosted (MINIO) endpoint
efi <- open_dataset("s3://neon4cast-scores/parquet/aquatics",
                    s3_access_key_id="",
                    s3_endpoint="data.ecoforecast.org")

# Use parser-options for non-standard csv:
 cars <- tempfile() # dummy data
 write.table(mtcars, cars, row.names = FALSE)

# Note nested quotes on parser option for delimiter:
 df <- open_dataset(cars, format = "csv",
                    parser_options = c(delim = "' '", header = TRUE))
```

---

spatial_join *spatial_join*

Description

spatial_join

Usage

```
spatial_join(
  x,
  y,
  by = c("st_intersects", "st_within", "st_dwithin", "st_touches", "st_contains",
    "st_containsproperly", "st_covers", "st_overlaps", "st_crosses", "st_equals",
     "st_disjoint"),
  args = "",
```

```
  join = "left",
  tblname = tmp_tbl_name(),
  conn = cached_connection()
)
```

Arguments

x	a duckdb table with a spatial geometry column called "geom"
y	a duckdb table with a spatial geometry column called "geom"
by	A spatial join function, see details.
args	additional arguments to join function (e.g. distance for st_dwithin)
join	JOIN type (left, right, inner, full)
tblname	name for the temporary view
conn	the duckdb connection (imputed by duckdbfs by default, must be shared across both tables)

Details

Possible spatial joins include:

Function	Description
st_intersects	Geometry A intersects with geometry B
st_disjoint	The complement of intersects
st_within	Geometry A is within geometry B (complement of contains)
st_dwithin	Geometries are within a specified distance, expressed in the same units as the coordinate reference syste
st_touches	Two polygons touch if the that have at least one point in common, even if their interiors do not touch.
st_contains	Geometry A entirely contains to geometry B. (complement of within)
st_containsproperly	stricter version of `st_contains` (boundary counts as external)
st_covers	geometry B is inside or on boundary of A. (A polygon covers a point on its boundary but does not conta
st_overlaps	geometry A intersects but does not completely contain geometry B
st_equals	geometry A is equal to geometry B
st_crosses	Lines or points in geometry A cross geometry B.

All though SQL is not case sensitive, this function expects only lower case names for "by" functions.

Value

a (lazy) view of the resulting table. Users can continue to operate on using dplyr operations and call to_st() to collect this as an sf object.

Examples

```
# note we can read in remote data in a variety of vector formats:
countries <-
paste0("/vsicurl/",
       "https://github.com/cboettig/duckdbfs/",
```

```
       "raw/spatial-read/inst/extdata/world.gpkg") |>
open_dataset(format = "sf")

cities <-
 paste0("/vsicurl/https://github.com/cboettig/duckdbfs/raw/",
        "spatial-read/inst/extdata/metro.fgb") |>
 open_dataset(format = "sf")

countries |>
  dplyr::filter(iso_a3 == "AUS") |>
  spatial_join(cities)
```

`st_read_meta` *read spatial metadata*

Description

At this time, reads a subset of spatial metadata. This is similar to what is reported by `ogrinfo -json`

Usage

```
st_read_meta(
  path,
  layer = 1L,
  tblname = tbl_name(path),
  conn = cached_connection(),
  ...
)
```

Arguments

`path`	URL or path to spatial data file
`layer`	layer number to read metadata for, defaults to first layer.
`tblname`	metadata will be stored as a view with this name, by default this is based on the name of the file.
`conn`	A connection to a database.
`...`	optional additional arguments passed to `duckdb_s3_config()`. Note these apply after those set by the URI notation and thus may be used to override or provide settings not supported in that format.

Value

A lazy `dplyr::tbl` object containing core spatial metadata such as projection information.

Examples

```
st_read_meta("https://github.com/duckdb/duckdb_spatial/raw/main/test/data/amsterdam_roads.fgb")
```

`to_geojson` *Write geojson using duckdb's native JSON writer*

Description

Write geojson using duckdb's native JSON writer

Usage

```
to_geojson(dataset, path, conn = cached_connection(), id_col = NULL)
```

Arguments

`dataset`	a remote tbl object from `open_dataset`, or an in-memory data.frame.
`path`	a local file path or S3 path with write credentials
`conn`	duckdbfs database connection
`id_col`	(deprecated). to_geojson() will preserve all atomic columns as properties.

`to_h3j` *Write H3 hexagon data out as an h3j-compliant JSON file NOTE: the column containing H3 hashes must be named* `hexid`

Description

Write H3 hexagon data out as an h3j-compliant JSON file NOTE: the column containing H3 hashes must be named `hexid`

Usage

```
to_h3j(dataset, path, conn = cached_connection())
```

Arguments

`dataset`	a remote tbl object from `open_dataset`, or an in-memory data.frame.
`path`	a local file path or S3 path with write credentials
`conn`	duckdbfs database connection

Examples

```
# example code
```

to_json *to_json write data out as a JSON object*

Description

to_json write data out as a JSON object

Usage

```
to_json(
  dataset,
  path,
  conn = cached_connection(),
  array = TRUE,
  options = NULL
)
```

Arguments

dataset	a remote tbl object from open_dataset, or an in-memory data.frame.
path	a local file path or S3 path with write credentials
conn	duckdbfs database connection
array	generate a JSON array?
options	additional options as a char string, see

to_sf *Convert output to sf object*

Description

Convert output to sf object

Usage

```
to_sf(x, crs = NA, conn = cached_connection())
```

Arguments

x	a remote duckdb tbl (from open_dataset) or dplyr-pipeline thereof.
crs	The coordinate reference system, any format understood by sf::st_crs.
conn	the connection object from the tbl. Takes a duckdb table (from open_dataset) or a dataset or dplyr pipline and returns an sf object. **Important**: the table must have a geometry column, which you will almost always have to create first. Note: to_sf() triggers collection into R. This function is suitable to use at the end of a dplyr pipeline that will subset the data. Using this function on a large dataset without filtering first may exceed available memory.

Value

an sf class object (in memory).

Examples

```
library(dplyr)
csv_file <- system.file("extdata/spatial-test.csv", package="duckdbfs")

# Note that we almost always must first create a `geometry` column, e.g.
# from lat/long columns using the `st_point` method.
sf <-
  open_dataset(csv_file, format = "csv") |>
  mutate(geom = ST_Point(longitude, latitude)) |>
  to_sf()

# We can use the full space of spatial operations, including spatial
# and normal dplyr filters.  All operations are translated into a
# spatial SQL query by `to_sf`:
open_dataset(csv_file, format = "csv") |>
  mutate(geom = ST_Point(longitude, latitude)) |>
  mutate(dist = ST_Distance(geom, ST_Point(0,0))) |>
  filter(site %in% c("a", "b", "e")) |>
  to_sf()
```

---

write_dataset *write_dataset*

Description

write_dataset

Usage

```
write_dataset(
  dataset,
  path,
  conn = cached_connection(),
  format = c("parquet", "csv"),
  partitioning = dplyr::group_vars(dataset),
  overwrite = TRUE,
  options = list(),
  ...
)
```

Arguments

dataset	a remote tbl object from open_dataset, or an in-memory data.frame.
path	a local file path or S3 path with write credentials
conn	duckdbfs database connection
format	export format
partitioning	names of columns to use as partition variables
overwrite	allow overwriting of existing files?
options	Additional arguments to COPY, see https://duckdb.org/docs/stable/sql/statements/copy.html#copy--to-options Note, uses duckdb native syntax, e.g. c("PER_THREAD_OUTPUT false"), for named arguments, see examples. (Recall SQL is case-insensitive).
...	additional arguments to duckdb_s3_config()

Value

Returns the path, invisibly.

Examples

```
  write_dataset(mtcars, tempfile())


write_dataset(mtcars, tempdir())
write_dataset(mtcars, tempdir(), options = c("PER_THREAD_OUTPUT FALSE", "RETURN_STATS TRUE"))
```

---

write_geo *Write a spatial file with gdal*

Description

Write a spatial file with gdal

Usage

```
write_geo(
  dataset,
  path,
  conn = cached_connection(),
  driver = "GeoJSON",
  layer_creation_options = "WRITE_BBOX=YES",
  srs = "ESPG:4326"
)
```

Arguments

dataset	a remote tbl object from open_dataset, or an in-memory data.frame.
path	a local file path or S3 path with write credentials
conn	duckdbfs database connection
driver	driver, see https://duckdb.org/docs/stable/extensions/spatial/gdal
layer_creation_options	to GDAL, see https://duckdb.org/docs/stable/extensions/spatial/gdal
srs	Set a spatial reference system as metadata to use for the export. This can be a WKT string, an EPSG code or a proj-string, basically anything you would normally be able to pass to GDAL. Note that this will not perform any reprojection of the input geometry, it just sets the metadata if the target driver supports it.

Details

NOTE: at this time, duckdb's pre-packaged GDAL does not support s3 writes, and will produce a "Error: Not implemented Error: GDAL Error (6): Seek not supported on writable /vsis3/ files". Use to_geojson() to export using duckdb's native JSON serializer instead.

Examples

```
local_file <-  system.file("extdata/spatial-test.csv", package="duckdbfs")
load_spatial()
tbl <- open_dataset(local_file, format='csv')
write_geo(tbl, "spatial.geojson")
```

Index