Package: mongolite (via r-universe)

October 28, 2026

Type Package

Title Fast and Simple 'MongoDB' Client for R

Description High-performance MongoDB client based on 'mongo-c-driver' and 'jsonlite'. Includes support for aggregation, indexing, map-reduce, streaming, encryption, enterprise authentication, and GridFS. The online user manual provides an overview of the available methods in the package: <https://jeroen.github.io/mongolite/>.

Version 4.1.0

Imports jsonlite (>= 1.4), openssl (>= 1.0), mime

License Apache License 2.0

URL https://jeroen.r-universe.dev/mongolite

BugReports https://github.com/jeroen/mongolite/issues

SystemRequirements OpenSSL, Cyrus SASL (aka libsasl2)

Suggests curl, spelling, nycflights13, ggplot2

Language en-GB

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Jeroen Ooms [aut, cre] (ORCID: <https://orcid.org/0000-0002-4035-0289>), MongoDB, Inc [cph] (Bundled mongo-c-driver, see AUTHORS file)

Maintainer Jeroen Ooms <jeroenooms@gmail.com>

Config/pak/sysreqs libssl-dev libsasl2-dev

Repository https://cran.r-universe.dev

Date/Publication 2026-07-20 20:45:35 UTC

RemoteUrl https://github.com/cran/mongolite

RemoteRef HEAD

RemoteSha 97f10a33dc40a8dad1e3dd565326cbaea230f7c3

Contents

gridfs 2
mongo 4
mongo_options 7
oid_to_timestamp 8
read_bson 8
ssl_options 9

Index **11**

`gridfs` *GridFS API*

Description

Connect to a GridFS database to search, read, write and delete files.

Usage

```
gridfs(
  db = "test",
  url = "mongodb://localhost",
  prefix = "fs",
  options = ssl_options()
)
```

Arguments

`db`	name of database
`url`	address of the mongodb server in mongo connection string URI format
`prefix`	string to prefix the collection name
`options`	additional connection options such as SSL keys/certs.

Details

We support two interfaces for sending/receiving data from/to GridFS. The `fs$read()` and `fs$write()` methods are the most flexible and can send data from/to an R connection, such as a file, socket or url. These methods support a progress counter and can be interrupted if needed. These methods are recommended for reading or writing single files.

The `fs$upload()` and `fs$download()` methods on the other hand copy directly between GridFS and your local disk. This API is vectorized so it can transfer many files at once. However individual transfers cannot be interrupted and will block R until completed. This API is only recommended to upload/download a large number of small files.

Modifying files in GridFS is currently unsupported: uploading a file with the same name will generate a new file.

Methods

`find(filter = "{}", options = "{}")` Search and list files in the GridFS

`download(name, path = '.')` Download one or more files from GridFS to disk. Path may be an existing directory or vector of filenames equal to 'name'.

`upload(path, name = basename(path), content_type = NULL, metadata = NULL)` Upload one or more files from disk to GridFS. Metadata is an optional JSON string.

`read(name, con = NULL, progress = TRUE)` Reads a single file from GridFS into a writable R connection. If con is a string it is treated as a filepath; if it is `NULL` then the output is buffered in memory and returned as a raw vector.

`write(con, name, content_type = NULL, metadata = NULL, progress = TRUE)` Stream write a single file into GridFS from a readable R connection. If con is a string it is treated as a filepath; it may also be a raw vector containing the data to upload. Metadata is an optional JSON string.

`remove(name)` Remove a single file from the GridFS

`drop()` Removes the entire GridFS collection, including all files

Examples

```
# Upload a file to GridFS
fs <- gridfs(url = "mongodb+srv://readwrite:test@cluster0-84vdt.mongodb.net/test")
input <- file.path(R.home('doc'), "html/logo.jpg")
fs$upload(input, name = 'logo.jpg')

# Download the file back to disk
output <- file.path(tempdir(), 'logo1.jpg')
fs$download('logo.jpg', output)

# Or you can also stream it
con <- file(file.path(tempdir(), 'logo2.jpg'))
fs$read('logo.jpg', con)

# Delete the file on the server
fs$remove('logo.jpg')

files <- c(input, file.path(tempdir(), c('logo1.jpg', 'logo2.jpg')))
hashes <- tools::md5sum(files)
stopifnot(length(unique(hashes)) == 1)

## Not run:
# Insert Binary Data
fs <- gridfs()
buf <- serialize(nycflights13::flights, NULL)
fs$write(buf, 'flights')
out <- fs$read('flights')
flights <- unserialize(out$data)

tmp <- file.path(tempdir(), 'flights.rds')
fs$download('flights', tmp)
flights2 <- readRDS(tmp)
```

```
stopifnot(all.equal(flights, nycflights13::flights))
stopifnot(all.equal(flights2, nycflights13::flights))

# Show what we have
fs$find()
fs$drop()

## End(Not run)
```

---

mongo *MongoDB client*

Description

Connect to a MongoDB collection. Returns a mongo connection object with methods listed below. Connections automatically get pooled between collection and gridfs objects to the same database.

Usage

```
mongo(
  collection = "test",
  db = "test",
  url = "mongodb://localhost",
  verbose = FALSE,
  options = ssl_options()
)
```

Arguments

collection	name of collection
db	name of database
url	address of the mongodb server in mongo connection string URI format
verbose	emit some more output
options	additional connection options such as SSL keys/certs.

Details

This manual page is deliberately minimal, see the mongolite user manual for more details and worked examples.

Value

Upon success returns a pointer to a collection on the server. The collection can be interfaced using the methods described below.

Methods

`aggregate(pipeline = '{}', handler = NULL, pagesize = 1000, iterate = FALSE)` Execute a pipeline using the Mongo aggregation framework. Set `iterate = TRUE` to return an iterator instead of data frame.

`count(query = '{}')` Count the number of records matching a given `query`. Default counts all records in collection.

`disconnect(gc = TRUE)` Disconnect collection. The *connection* gets disconnected once the client is not used by collections in the pool.

`distinct(key, query = '{}')` List unique values of a field given a particular query.

`drop()` Delete entire collection with all data and metadata.

`export(con = stdout(), bson = FALSE, query = '{}', fields = '{}', sort = '{"_id":1}')` Streams all data from collection to a `connection` in jsonlines format (similar to mongoexport). Alternatively when `bson = TRUE` it outputs the binary bson format (similar to mongodump).

`find(query = '{}', fields = '{"_id" : 0}', sort = '{}', skip = 0, limit = 0, handler = NULL, pagesize = 1000)` Retrieve `fields` from records matching `query`. Default `handler` will return all data as a single dataframe.

`import(con, bson = FALSE)` Stream import data in jsonlines format from a `connection`, similar to the mongoimport utility. Alternatively when `bson = TRUE` it assumes the binary bson format (similar to mongorestore).

`index(add = NULL, remove = NULL)` List, add, or remove indexes from the collection. The `add` and `remove` arguments can either be a field name or json object. Returns a dataframe with current indexes.

`info()` Returns collection statistics and server info (if available).

`insert(data, pagesize = 1000, stop_on_error = TRUE, ...)` Insert rows into the collection. Argument 'data' must be a data-frame, named list (for single record) or character vector with json strings (one string for each row). For lists and data frames, arguments in `...` get passed to `jsonlite::toJSON`

`iterate(query = '{}', fields = '{"_id":0}', sort = '{}', skip = 0, limit = 0)` Runs query and returns iterator to read single records one-by-one.

`mapreduce(map, reduce, query = '{}', sort = '{}', limit = 0, out = NULL, scope = NULL)` Performs a map reduce query. The `map` and `reduce` arguments are strings containing a JavaScript function. Set `out` to a string to store results in a collection instead of returning.

`remove(query = "{}", just_one = FALSE)` Remove record(s) matching `query` from the collection.

`rename(name, db = NULL)` Change the name or database of a collection. Changing name is cheap, changing database is expensive.

`replace(query, update = '{}', upsert = FALSE)` Replace matching record(s) with value of the `update` argument.

`run(command = '{"ping": 1}', simplify = TRUE)` Run a raw mongodb command on the database. If the command returns data, output is simplified by default, but this can be disabled.

`update(query, update = '{"$set":{}}', upsert = FALSE, multiple = FALSE)` Modify fields of matching record(s) with value of the `update` argument.

References

Mongolite User Manual

Jeroen Ooms (2014). The jsonlite Package: A Practical and Consistent Mapping Between JSON Data and R Objects. *arXiv:1403.2805*. https://arxiv.org/abs/1403.2805

Examples

```
# Connect to demo server
con <- mongo("mtcars", url =
  "mongodb+srv://readwrite:test@cluster0-84vdt.mongodb.net/test")
if(con$count() > 0) con$drop()
con$insert(mtcars)
stopifnot(con$count() == nrow(mtcars))

# Query data
mydata <- con$find()
stopifnot(all.equal(mydata, mtcars))
con$drop()

# Automatically disconnect when connection is removed
rm(con)
gc()

## Not run:
# dplyr example
library(nycflights13)

# Insert some data
m <- mongo(collection = "nycflights")
m$drop()
m$insert(flights)

# Basic queries
m$count('{"month":1, "day":1}')
jan1 <- m$find('{"month":1, "day":1}')

# Sorting
jan1 <- m$find('{"month":1,"day":1}', sort='{"distance":-1}')
head(jan1)

# Sorting on large data requires index
m$index(add = "distance")
allflights <- m$find(sort='{"distance":-1}')

# Select columns
jan1 <- m$find('{"month":1,"day":1}', fields = '{"_id":0, "distance":1, "carrier":1}')

# List unique values
m$distinct("carrier")
m$distinct("carrier", '{"distance":{"$gt":3000}}')

# Tabulate
```

```
m$aggregate('[{"$group":{"_id":"$carrier", "count": {"$sum":1}, "average":{"$avg":"$distance"}}}]')

# Map-reduce (binning)
hist <- m$mapreduce(
  map = "function(){emit(Math.floor(this.distance/100)*100, 1)}",
  reduce = "function(id, counts){return Array.sum(counts)}"
)

# Stream jsonlines into a connection
tmp <- tempfile()
m$export(file(tmp))

# Remove the collection
m$drop()

# Import from jsonlines stream from connection
dmd <- mongo("diamonds")
dmd$import(url("http://jeroen.github.io/data/diamonds.json"))
dmd$count()

# Export
dmd$drop()

## End(Not run)
```

`mongo_options` *Mongo Options*

Description

Get and set global client options. Calling with `NULL` parameters returns current values without modifying.

Usage

```
mongo_options(log_level = NULL, bigint_as_char = NULL, date_as_char = NULL)
```

Arguments

`log_level`	integer between 0 and 6 or `NULL` to leave unchanged.
`bigint_as_char`	logical: parse int64 as strings instead of double.
`date_as_char`	logical: parse UTC datetime as strings instead of POSIXct.

Details

Setting log_level to 0 suppresses critical warnings and messages, while 6 is most verbose and displays all debugging information. Possible values for level are:

- `0`: *error*

- 1: *critical*
- 2: *warning*
- 3: *message*
- 4: *info* (**default**)
- 5: *debug*
- 6: *trace*

Note that setting it below 2 will suppress important warnings and setting below 1 will suppress critical errors (not recommended). The default is 4.

`oid_to_timestamp` *Get OID date*

Description

The initial 4 bytes of a MongoDB OID contain a timestamp value, representing the ObjectId creation, measured in seconds since the Unix epoch.

Usage

```
oid_to_timestamp(oid)
```

Arguments

`oid`	string or raw value with document oid

Examples

```
oid_to_timestamp('5349b4ddd2781d08c09890f3')
```

`read_bson` *Standalone BSON reader*

Description

Reads BSON data from a `mongoexport` dump file directly into R (if it can fit in memory). This utility does not attempt to convert result into one big single data.frame: the output is always a vector of length equal to total number of documents in the collection.

Usage

```
read_bson(file, as_json = FALSE, simplify = TRUE, verbose = interactive())
```

Arguments

file	path or url to a bson file
as_json	read data into json strings instead of R lists.
simplify	should nested data get simplified into atomic vectors and dataframes where possible? Only used for as_json = FALSE.
verbose	print some progress output while reading

Details

It is enabled by default to simplify the individual data documents using the same rules as jsonlite. This converts nested lists into atomic vectors and data frames when possible, which makes data easier to work with in R.

An alternative to this function is to import your BSON file into a local mongodb server using the mongo$import() function. This requires little memory and once data is in mongodb you can easily query and modify it.

Examples

```
diamonds <- read_bson("https://jeroen.github.io/data/diamonds.bson")
length(diamonds)
```

---

ssl_options *Connection SSL options*

Description

Set SSL options to connect to the MongoDB server.

Usage

```
ssl_options(
  cert = NULL,
  key = cert,
  ca = NULL,
  ca_dir = NULL,
  crl_file = NULL,
  allow_invalid_hostname = NULL,
  weak_cert_validation = NULL
)
```

Arguments

cert	path to PEM file with client certificate, or a certificate as returned by openssl::read_cert()
key	path to PEM file with private key from the above certificate, or a key as returned by openssl::read_key(). This can be the same PEM file as cert.

`ca`	a certificate authority PEM file
`ca_dir`	directory with CA files
`crl_file`	file with revocations
`allow_invalid_hostname`	do not verify hostname on server certificate
`weak_cert_validation`	disable certificate verification

Index

connection, *3*, *5*

file, *2*

gridfs, 2

jsonlite, *9*
jsonlite::toJSON, *5*

mongo, 4, *4*
mongo$import(), *9*
mongo_options, 7
mongolite *(*mongo*)*, 4

oid_to_timestamp, 8
openssl::read_cert(), *9*
openssl::read_key(), *9*

raw, *3*
read_bson, 8

socket, *2*
ssl_options, 9

url, *2*