GridFS

On this page

- Creating a GridFS object (“Grid::FSBucket”)
- Working with write streams
- Working with read streams
- Finding file metadata
- Deleting files
- Working with Grid::File objects
- Inserting Files
- Finding Files
- Deleting Files

The driver provides a clean and simple interface to work with storage of chunked files in the database, also known as the pattern “GridFS”. The API allows you to either work with Grid::File objects or with read and write streams.

Creating a GridFS object (“Grid::FSBucket”)

You can create a GridFS object by calling `fs` on a database, with optional arguments. `fs` returns a `Grid::FSBucket` object.

The options that `Grid::FSBucket` supports are:

Option	Description
`:bucket_name`	The name of the GridFS Bucket. Default is `fs`.
`:fs_name`	The name of the GridFS Bucket. Takes precedence over `bucket_name`. Default is `fs`.

Option	Description
`:chunk_size`	Specifies the size of each file chunk in the database.
`:write`	The write concern used when uploading files.
`:read`	The read preference used when downloading files.

For example, you can create a GridFS bucket object with a particular read preference:

```
fs_bucket = database.fs( read: { mode: :secondary } )
```

Working with write streams

To upload a file to GridFS using a write stream, you can either open a stream and write to it directly or write the entire contents of an IO object to GridFS all at once.

To open an upload stream and write to it:

```
file = File.open('/path/to/my-file.txt', 'r')
fs_bucket.open_upload_stream('my-file.txt') do |stream|
  stream.write(file)
end
file.close
```

To upload the entire contents of an IO object in one call:

```
file = File.open('/path/to/my-file.txt', 'r')
fs_bucket.upload_from_stream('my-file.txt', file)
file.close
```

Working with read streams

To download a file from GridFS using a read stream, you can either open a read stream and read from it directly or download the entire file all at once.

To open a download stream and read from it:

```
file = File.open('/path/to/my-output-file.txt', 'w')
fs_bucket.open_download_stream(file_id) do |stream|
  file.write(stream.read)
end
file.close
```

To download the file all at once and write it to an IO object:

```
file = File.open('/path/to/my-output-file.txt', 'w')
fs_bucket.download_from_stream(file_id, file)
file.close
```

You can also download a file specified by a name and (optionally) revision number. Revision numbers are used to distinguish between files sharing the same name, ordered by date of upload. The revision number passed to `open_download_stream_by_name` can be positive or negative.

```
file = File.open('/path/to/my-output-file.txt', 'w')
fs_bucket.open_download_stream_by_name('my-file.txt', revision: -2) do |stream|
  file.write(stream.read)
end
file.close
```

To download the entire contents of the file specified by name and (optionally) revision number:

```
file = File.open('/path/to/my-output-file.txt', 'w')
fs_bucket.download_to_stream_by_name('my-file.txt', file, revision: -2)
file.close
```

Finding file metadata

You can retrieve documents containing metadata about files in the GridFS files collection.

```
fs_bucket.find(filename: 'my-file.txt')
```

Deleting files

You can delete a file by id.

```
fs_bucket.delete(file_id)
```

Working with Grid::File objects

This object can be used to wrap a file to be inserted into the database using GridFS and the object that is retrieved.

To create a file with raw data:

```
file = Mongo::Grid::File.new('I am a file', :filename => 'new-file.txt')
```

To create a file from a Ruby `File` object:

```
file = File.open('/path/to/my-file.txt')
grid_file = Mongo::Grid::File.new(file.read, :filename => File.basename(file.path))
```

To change file options such as chunk size, pass options to the constructor:

```
file = File.open('/path/to/my-file.txt')
grid_file = Mongo::Grid::File.new(
  file.read,
  :filename => File.basename(file.path),
  :chunk_size => 1024
)
```

The following is a full list of the available options that files support.

Option	Description
`:chunk_size`	Sets the size of each file chunk in the database.
`:content_type`	Set a content type for the file.
`:filename` (Required)	The file name.
`:upload_date`	The date the file was uploaded (stored).

Inserting Files

Files can be inserted into the database one at a time. File chunks are inserted by default into the `fs.chunks` collection and file metadata is inserted into the `fs.files` collection.

```
client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'music')
file = Mongo::Grid::File.new('I am a file', :filename => 'new-file.txt')

client.database.fs.insert_one(file)
```

To insert into collections with a name prefix other than fs, access the filesystem with a `:fs_name` option.

```
client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'music')
file = Mongo::Grid::File.new('I am a file', :filename => 'new-file.txt')

client.database.fs(:fs_name => 'grid').insert_one(file)
```

Note that the first time a file is inserted, it will create the required index for you on the `chunks` collection. This index is a compound index:

```
{ :files_id => 1, :n => 1 }
```

Files can also be streamed as an alternative to a direct insert.

```
client.database.fs.open_upload_stream(filename) do |stream|
  stream.write(file)
end
```

Finding Files

To retrieve a file from the database, call `find_one` with the appropriate filter.

```
client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'music')
client.database.fs.find_one(:filename => 'new-file.txt') # Returns a Mongo::Grid::File
```

Files can also be streamed as an alternative to a direct find.

```
client.database.fs.open_download_stream(file_id) do |stream|
  io.write(stream.read)
end

fs.download_to_stream(file_id, io)
```

Deleting Files

To delete a file, pass the file object to `delete_one`.

```
client = Mongo::Client.new([ '127.0.0.1:27017' ], :database => 'music')
fs = client.database.fs
file = fs.find_one(:filename => 'new-file.txt')
fs.delete_one(file)


  coll.find.to_a # { '_id' => ..., 'x' => 3 }
```