# Mongoose

## Mongoose()

#### Parameters

- options «Object» see `Mongoose#set()` docs

Mongoose constructor.

The exports object of the `mongoose` module is an instance of this class. Most apps will only use this one instance.

### Example:

```
const mongoose = require('mongoose');
mongoose instanceof mongoose.Mongoose; // true

// Create a new Mongoose instance with its own `connect()`, `set()`, `model()`, etc.
const m = new mongoose.Mongoose();
```

## Mongoose.prototype.Aggregate()

The Mongoose Aggregate constructor

## Mongoose.prototype.CastError()

Parameters

- type «String» The name of the type
- value «Any» The value that failed to cast
- path «String» The path `a.b.c` in the doc where this cast error occurred
- [reason] «Error» The original error that was thrown

The Mongoose CastError constructor

## Mongoose.prototype.Collection()

The Mongoose Collection constructor

## Mongoose.prototype.Connection()

The Mongoose Connection constructor

## Mongoose.prototype.Date

Type:

- «property»

The Mongoose Date SchemaType.

**Example:**

```
const schema = new Schema({ test: Date });
schema.path('test') instanceof mongoose.Date; // true
```

## Mongoose.prototype.Decimal128

Type:

- «property»

The Mongoose Decimal128 SchemaType. Used for declaring paths in your schema that should be 128-bit decimal floating points. Do not use this to create a new Decimal128 instance, use `mongoose.Types.Decimal128` instead.

**Example:**

```
const vehicleSchema = new Schema({ fuelLevel: mongoose.Decimal128 });
```

## Mongoose.prototype.Document()

The Mongoose Document constructor.

## Mongoose.prototype.DocumentProvider()

The Mongoose DocumentProvider constructor. Mongoose users should not have to use this directly

## Mongoose.prototype.Error()

The MongooseError constructor.

## Mongoose.prototype.Mixed

**Type:**

- «property»

The Mongoose Mixed SchemaType. Used for declaring paths in your schema that Mongoose's change tracking, casting, and validation should ignore.

**Example:**

```
const schema = new Schema({ arbitrary: mongoose.Mixed });
```

## Mongoose.prototype.Model()

The Mongoose Model constructor.

## Mongoose.prototype.Mongoose()

The Mongoose constructor

The exports of the mongoose module is an instance of this class.

### Example:

```
const mongoose = require('mongoose');
const mongoose2 = new mongoose.Mongoose();
```

## Mongoose.prototype.Number

#### Type:

- «property»

The Mongoose Number SchemaType. Used for declaring paths in your schema that Mongoose should cast to numbers.

### Example:

```
const schema = new Schema({ num: mongoose.Number });
// Equivalent to:
const schema = new Schema({ num: 'number' });
```

## Mongoose.prototype.ObjectId

#### Type:

- «property»

The Mongoose ObjectId SchemaType. Used for declaring paths in your schema that should be MongoDB ObjectIds. Do not use this to create a new ObjectId instance, use `mongoose.Types.ObjectId` instead.

### Example:

```
const childSchema = new Schema({ parentId: mongoose.ObjectId });
```

## Mongoose.prototype.Promise

**Type:**

- «property»

The Mongoose Promise constructor.

## Mongoose.prototype.PromiseProvider()

Storage layer for mongoose promises

## Mongoose.prototype.Query()

The Mongoose Query constructor.

## Mongoose.prototype.STATES

**Type:**

- «property»

Expose connection states for user-land

## Mongoose.prototype.Schema()

The Mongoose Schema constructor

### Example:

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;
const CatSchema = new Schema(..);
```

## Mongoose.prototype.SchemaType()

The Mongoose SchemaType constructor

## Mongoose.prototype.SchemaTypeOptions()

The constructor used for schematype options

## Mongoose.prototype.SchemaTypes

**Type:**

- «property»

The various Mongoose SchemaTypes.

**Note:**

*Alias of mongoose.Schema.Types for backwards compatibility.*

## Mongoose.prototype.Types

**Type:**

- «property»

The various Mongoose Types.

**Example:**

```
const mongoose = require('mongoose');
const array = mongoose.Types.Array;
```

**Types:**

- Array
- Buffer
- Embedded
- DocumentArray
- Decimal128
- ObjectId
- Map
- Subdocument

Using this exposed access to the `ObjectId` type, we can construct ids on demand.

```
const ObjectId = mongoose.Types.ObjectId;
const id1 = new ObjectId;
```

## Mongoose.prototype.VirtualType()

The Mongoose VirtualType constructor

## Mongoose.prototype.connect()

**Parameters**

- uri(s) «String»
- [options] «Object» passed down to the MongoDB driver's `connect()` function, except for 4 mongoose-specific options explained below.
  - [options.bufferCommands=true] «Boolean» Mongoose specific option. Set to false to disable buffering on all models associated with this connection.
  - [options.bufferTimeoutMS=10000] «Number» Mongoose specific option. If `bufferCommands` is true, Mongoose will throw an error after `bufferTimeoutMS` if the operation is still buffered.
  - [options.dbName] «String» The name of the database we want to use. If not provided, use database name from connection string.
  - [options.user] «String» username for authentication, equivalent to `options.auth.user`. Maintained for backwards compatibility.
  - [options.pass] «String» password for authentication, equivalent to `options.auth.password`. Maintained for backwards compatibility.
  - [options.maxPoolSize=100] «Number» The maximum number of sockets the MongoDB driver will keep open for this connection. Keep in mind that MongoDB only allows one operation per socket at a time, so you may want to increase this if you find you have a few slow queries that are blocking faster queries from proceeding. See Slow Trains in MongoDB and Node.js.
  - [options.minPoolSize=0] «Number» The minimum number of sockets the MongoDB driver will keep open for this connection.
  - [options.serverSelectionTimeoutMS] «Number» If `useUnifiedTopology = true`, the MongoDB driver will try to find a server to send any given operation to, and keep retrying for `serverSelectionTimeoutMS` milliseconds before erroring out. If not set, the MongoDB driver defaults to using `30000` (30 seconds).
  - [options.heartbeatFrequencyMS] «Number» If `useUnifiedTopology = true`, the MongoDB driver sends a heartbeat every `heartbeatFrequencyMS` to check on the status of the

connection. A heartbeat is subject to `serverSelectionTimeoutMS`, so the MongoDB driver will retry failed heartbeats for up to 30 seconds by default. Mongoose only emits a `'disconnected'` event after a heartbeat has failed, so you may want to decrease this setting to reduce the time between when your server goes down and when Mongoose emits `'disconnected'`. We recommend you do **not** set this setting below 1000, too many heartbeats can lead to performance degradation.

  - [options.autoIndex=true] «Boolean» Mongoose-specific option. Set to false to disable automatic index creation for all models associated with this connection.
  - [options.reconnectTries=30] «Number» If you're connected to a single server or mongos proxy (as opposed to a replica set), the MongoDB driver will try to reconnect every `reconnectInterval` milliseconds for `reconnectTries` times, and give up afterward. When the driver gives up, the mongoose connection emits a `reconnectFailed` event. This option does nothing for replica set connections.
  - [options.reconnectInterval=1000] «Number» See `reconnectTries` option above.
  - [options.promiseLibrary] «Class» Sets the underlying driver's promise library.
  - [options.connectTimeoutMS=30000] «Number» How long the MongoDB driver will wait before killing a socket due to inactivity *during initial connection*. Defaults to 30000. This option is passed transparently to Node.js' `socket#setTimeout()` function.
  - [options.socketTimeoutMS=30000] «Number» How long the MongoDB driver will wait before killing a socket due to inactivity *after initial connection*. A socket may be inactive because of either no activity or a long-running operation. This is set to `30000` by default, you should set this to 2-3x your longest running operation if you expect some of your database operations to run longer than 20 seconds. This option is passed to Node.js `socket#setTimeout()` function after the MongoDB driver successfully completes.
  - [options.family=0] «Number» Passed transparently to Node.js' `dns.lookup()` function. May be either `0`, `4`, or `6`. `4` means use IPv4 only, `6` means use IPv6 only, `0` means try both.
  - [options.autoCreate=false] «Boolean» Set to `true` to make Mongoose automatically call `createCollection()` on every model created on this connection.

- [callback] «Function»

**Returns:**

- «Promise» resolves to `this` if connection succeeded

Opens the default mongoose connection.

**Example:**

```
mongoose.connect('mongodb://user:pass@localhost:port/database');

// replica sets
const uri = 'mongodb://user:pass@localhost:port,anotherhost:port,yetanother:port/mydatabase';
mongoose.connect(uri);

// with options
```

```
mongoose.connect(uri, options);

// optional callback that gets fired when initial connection completed
const uri = 'mongodb://nonexistent.domain:27000';
mongoose.connect(uri, function(error) {
  // if error is truthy, the initial connection failed.
})
```

## Mongoose.prototype.connection

**Type:**

- «Connection»

The Mongoose module's default connection. Equivalent to `mongoose.connections[0]`, see `connections`.

### Example:

```
const mongoose = require('mongoose');
mongoose.connect(...);
mongoose.connection.on('error', cb);
```

This is the connection used by default for every model created using mongoose.model.

To create a new connection, use `createConnection()`.

## Mongoose.prototype.connections

**Type:**

- «Array»

An array containing all connections associated with this Mongoose instance. By default, there is 1 connection. Calling `createConnection()` adds a connection to this array.

### Example:

```
const mongoose = require('mongoose');
mongoose.connections.length; // 1, just the default connection
mongoose.connections[0] === mongoose.connection; // true

mongoose.createConnection('mongodb://localhost:27017/test');
mongoose.connections.length; // 2
```

## Mongoose.prototype.createConnection()

Parameters

- [uri] «String» a mongodb:// URI
- [options] «Object» passed down to the MongoDB driver's `connect()` function, except for 4 mongoose-specific options explained below.
    - [options.bufferCommands=true] «Boolean» Mongoose specific option. Set to false to disable buffering on all models associated with this connection.
    - [options.dbName] «String» The name of the database you want to use. If not provided, Mongoose uses the database name from connection string.
    - [options.user] «String» username for authentication, equivalent to `options.auth.user`. Maintained for backwards compatibility.
    - [options.pass] «String» password for authentication, equivalent to `options.auth.password`. Maintained for backwards compatibility.
    - [options.autoIndex=true] «Boolean» Mongoose-specific option. Set to false to disable automatic index creation for all models associated with this connection.
    - [options.reconnectTries=30] «Number» If you're connected to a single server or mongos proxy (as opposed to a replica set), the MongoDB driver will try to reconnect every `reconnectInterval` milliseconds for `reconnectTries` times, and give up afterward. When the driver gives up, the mongoose connection emits a `reconnectFailed` event. This option does nothing for replica set connections.
    - [options.reconnectInterval=1000] «Number» See `reconnectTries` option above.
    - [options.promiseLibrary] «Class» Sets the underlying driver's promise library.
    - [options.maxPoolSize=5] «Number» The maximum number of sockets the MongoDB driver will keep open for this connection. Keep in mind that MongoDB only allows one operation per socket at a time, so you may want to increase this if you find you have a few slow queries that are blocking faster queries from proceeding. See Slow Trains in MongoDB and Node.js.
    - [options.minPoolSize=1] «Number» The minimum number of sockets the MongoDB driver will keep open for this connection. Keep in mind that MongoDB only allows one operation per socket at a time, so you may want to increase this if you find you have a few slow queries that are blocking faster queries from proceeding. See Slow Trains in MongoDB and Node.js.
    - [options.connectTimeoutMS=30000] «Number» How long the MongoDB driver will wait before killing a socket due to inactivity *during initial connection*. Defaults to 30000. This option is passed transparently to Node.js' `socket#setTimeout()` function.
    - [options.socketTimeoutMS=30000] «Number» How long the MongoDB driver will wait before killing a socket due to inactivity *after initial connection*. A socket may be inactive because of either no activity or a long-running operation. This is set to `30000` by default, you should set this to 2-3x your longest running operation if you expect some of your database operations to run longer than 20 seconds. This option is passed to Node.js `socket#setTimeout()` function after the MongoDB driver successfully completes.

    - [options.family=0] «Number» Passed transparently to Node.js' `dns.lookup()` function. May be either `0`, `4`, or `6`. `4` means use IPv4 only, `6` means use IPv6 only, `0` means try both.

**Returns:**

- «Connection» the created Connection object. Connections are thenable, so you can do `await mongoose.createConnection()`

Creates a Connection instance.

Each `connection` instance maps to a single database. This method is helpful when managing multiple db connections.

*Options passed take precedence over options included in connection strings.*

### Example:

```
// with mongodb:// URI
db = mongoose.createConnection('mongodb://user:pass@localhost:port/database');

// and options
const opts = { db: { native_parser: true }}
db = mongoose.createConnection('mongodb://user:pass@localhost:port/database', opts);

// replica sets
db = mongoose.createConnection('mongodb://user:pass@localhost:port,anotherhost:port,yetanother:por

// and options
const opts = { replset: { strategy: 'ping', rs_name: 'testSet' }}
db = mongoose.createConnection('mongodb://user:pass@localhost:port,anotherhost:port,yetanother:por

// and options
const opts = { server: { auto_reconnect: false }, user: 'username', pass: 'mypassword' }
db = mongoose.createConnection('localhost', 'database', port, opts)

// initialize now, connect later
db = mongoose.createConnection();
db.openUri('localhost', 'database', port, [opts]);
```

## Mongoose.prototype.deleteModel()

**Parameters**

- name «String|RegExp» if string, the name of the model to remove. If regexp, removes all models whose name matches the regexp.

**Returns:**

- «Mongoose» this

Removes the model named `name` from the default connection, if it exists. You can use this function to clean up any models you created in your tests to prevent OverwriteModelErrors.

Equivalent to `mongoose.connection.deleteModel(name)`.

**Example:**

```
mongoose.model('User', new Schema({ name: String }));
console.log(mongoose.model('User')); // Model object
mongoose.deleteModel('User');
console.log(mongoose.model('User')); // undefined

// Usually useful in a Mocha `afterEach()` hook
afterEach(function() {
  mongoose.deleteModel(/.+/); // Delete every model
});
```

## Mongoose.prototype.disconnect()

**Parameters**

- [callback] «Function» called after all connection close, or when first error occurred.

**Returns:**

- «Promise» resolves when all connections are closed, or rejects with the first error that occurred.

Runs `.close()` on all connections in parallel.

## Mongoose.prototype.driver

**Type:**

- «property»

Object with `get()` and `set()` containing the underlying driver this Mongoose instance uses to communicate with the database. A driver is a Mongoose-specific interface that defines functions like `find()`.

## Mongoose.prototype.get()

**Parameters**

- key «String»

Gets mongoose options

**Example:**

```
mongoose.get('test') // returns the 'test' value
```

## Mongoose.prototype.isValidObjectId()

Returns true if Mongoose can cast the given value to an ObjectId, or false otherwise.

### Example:

```
mongoose.isValidObjectId(new mongoose.Types.ObjectId()); // true
mongoose.isValidObjectId('0123456789ab'); // true
mongoose.isValidObjectId(6); // false
```

## Mongoose.prototype.model()

### Parameters

- name «String|Function» model name or class extending Model
- [schema] «Schema» the schema to use.
- [collection] «String» name (optional, inferred from model name)

### Returns:

- «Model» The model associated with `name`. Mongoose will create the model if it doesn't already exist.

Defines a model or retrieves it.

Models defined on the `mongoose` instance are available to all connection created by the same `mongoose` instance.

If you call `mongoose.model()` with twice the same name but a different schema, you will get an `OverwriteModelError`. If you call `mongoose.model()` with the same name and same schema, you'll get the same schema back.

### Example:

```
const mongoose = require('mongoose');

// define an Actor model with this mongoose instance
const schema = new Schema({ name: String });
mongoose.model('Actor', schema);

// create a new connection
const conn = mongoose.createConnection(..);
```

```
// create Actor model
const Actor = conn.model('Actor', schema);
conn.model('Actor') === Actor; // true
conn.model('Actor', schema) === Actor; // true, same schema
conn.model('Actor', schema, 'actors') === Actor; // true, same schema and collection name

// This throws an `OverwriteModelError` because the schema is different.
conn.model('Actor', new Schema({ name: String }));
```

*When no `collection` argument is passed, Mongoose uses the model name. If you don't like this behavior, either pass a collection name, use `mongoose.pluralize()`, or set your schemas collection name option.*

**Example:**

```
const schema = new Schema({ name: String }, { collection: 'actor' });

// or

schema.set('collection', 'actor');

// or

const collectionName = 'actor'
const M = mongoose.model('Actor', schema, collectionName)
```

## Mongoose.prototype.modelNames()

**Returns:**

- «Array»

Returns an array of model names created on this instance of Mongoose.

**Note:**

*Does not include names of models created using `connection.model()`.*

## Mongoose.prototype.mongo

**Type:**

- «property»

The node-mongodb-native driver Mongoose uses.

## Mongoose.prototype.mquery

Type:

- «property»

The mquery query builder Mongoose uses.

## Mongoose.prototype.now()

Mongoose uses this function to get the current time when setting timestamps. You may stub out this function using a tool like Sinon for testing.

## Mongoose.prototype.plugin()

Parameters

- fn «Function» plugin callback
- [opts] «Object» optional options

Returns:

- «Mongoose» this

Declares a global plugin executed on all Schemas.

Equivalent to calling `.plugin(fn)` on each Schema you create.

## Mongoose.prototype.pluralize()

Parameters

- [fn] «Function|null» overwrites the function used to pluralize collection names

Returns:

- «Function,null» the current function used to pluralize collection names, defaults to the legacy function from `mongoose-legacy-pluralize`.

Getter/setter around function for pluralizing collection names.

## Mongoose.prototype.sanitizeFilter()

Parameters

- filter «Object»

Sanitizes query filters against query selector injection attacks by wrapping any nested objects that have a property whose name starts with `$` in a `$eq`.

```
const obj = { username: 'val', pwd: { $ne: null } };
sanitizeFilter(obj);
obj; // { username: 'val', pwd: { $eq: { $ne: null } } });
```

## Mongoose.prototype.set()

Parameters

- key «String»
- value «String|Function|Boolean»

Sets mongoose options

**Example:**

```
mongoose.set('test', value) // sets the 'test' option to `value`

mongoose.set('debug', true) // enable logging collection methods + arguments to the console/file

mongoose.set('debug', function(collectionName, methodName, ...methodArgs) {}); // use custom funct
```

## Currently supported options are

- 'debug': If `true`, prints the operations mongoose sends to MongoDB to the console. If a writable stream is passed, it will log to that stream, without colorization. If a callback function is passed, it will receive the collection name, the method name, then all arugments passed to the method. For example, if you wanted to replicate the default logging, you could output from the callback `Mongoose: ${collectionName}.${methodName}(${methodArgs.join(', ')})`.
- 'returnOriginal': If `false`, changes the default `returnOriginal` option to `findOneAndUpdate()`, `findByIdAndUpdate`, and `findOneAndReplace()` to false. This is equivalent to setting the `new` option to `true` for `findOneAndX()` calls by default. Read our `findOneAndUpdate()` tutorial for more information.
- 'bufferCommands': enable/disable mongoose's buffering mechanism for all connections and models
- 'cloneSchemas': false by default. Set to `true` to `clone()` all schemas before compiling into a model.
- 'applyPluginsToDiscriminators': false by default. Set to true to apply global plugins to discriminator schemas. This typically isn't necessary because plugins are applied to the base schema and discriminators copy all middleware, methods, statics, and properties from the base schema.

- 'applyPluginsToChildSchemas': true by default. Set to false to skip applying global plugins to child schemas
- 'objectIdGetter': true by default. Mongoose adds a getter to MongoDB ObjectId's called `_id` that returns `this` for convenience with populate. Set this to false to remove the getter.
- 'runValidators': false by default. Set to true to enable update validators for all validators by default.
- 'toObject': `{ transform: true, flattenDecimals: true }` by default. Overwrites default objects to `toObject()`
- 'toJSON': `{ transform: true, flattenDecimals: true }` by default. Overwrites default objects to `toJSON()`, for determining how Mongoose documents get serialized by `JSON.stringify()`
- 'strict': true by default, may be `false`, `true`, or `'throw'`. Sets the default strict mode for schemas.
- 'strictQuery': same value as 'strict' by default (`true`), may be `false`, `true`, or `'throw'`. Sets the default strictQuery mode for schemas.
- 'selectPopulatedPaths': true by default. Set to false to opt out of Mongoose adding all fields that you `populate()` to your `select()`. The schema-level option `selectPopulatedPaths` overwrites this one.
- 'maxTimeMS': If set, attaches maxTimeMS to every query
- 'autoIndex': true by default. Set to false to disable automatic index creation for all models associated with this Mongoose instance.
- 'autoCreate': Set to `true` to make Mongoose call `Model.createCollection()` automatically when you create a model with `mongoose.model()` or `conn.model()`. This is useful for testing transactions, change streams, and other features that require the collection to exist.
- 'overwriteModels': Set to `true` to default to overwriting models with the same name when calling `mongoose.model()`, as opposed to throwing an `OverwriteModelError`.

## Mongoose.prototype.startSession()

**Parameters**

- [options] «Object» see the mongodb driver options
  - [options.causalConsistency=true] «Boolean» set to false to disable causal consistency
- [callback] «Function»

**Returns:**

- «Promise<ClientSession>» promise that resolves to a MongoDB driver `ClientSession`

*Requires MongoDB >= 3.6.0.* Starts a MongoDB session for benefits like causal consistency, retryable writes, and transactions.

Calling `mongoose.startSession()` is equivalent to calling `mongoose.connection.startSession()`. Sessions are scoped to a connection, so calling `mongoose.startSession()` starts a session on the default mongoose connection.

## Mongoose.prototype.trusted()

#### Parameters

- obj «Object»

Tells `sanitizeFilter()` to skip the given object when filtering out potential query selector injection attacks. Use this method when you have a known query selector that you want to use.

```
const obj = { username: 'val', pwd: trusted({ $type: 'string', $eq: 'my secret' }) };
sanitizeFilter(obj);

// Note that `sanitizeFilter()` did not add `$eq` around `$type`.
obj; // { username: 'val', pwd: { $type: 'string', $eq: 'my secret' } });
```

## Mongoose.prototype.version

#### Type:

- «property»

The Mongoose version

### Example

```
console.log(mongoose.version); // '5.x.x'
```