\_Please note!\_ The github issue tracker should only be used for feature requests and bugs with a clear description of the issue and the expected behaviour (see below). All questions belong on [Slack](https://sequelize.slack.com) & [StackOverflow](https://stackoverflow.com/questions/tagged/sequelize.js).

# Issues

Issues are always very welcome - after all, they are a big part of making sequelize better. However, there are a couple of things you can do to make the lives of the developers \_much, much\_ easier:

### Tell us:

- What you are doing?

- Post a \_minimal\_ code sample that reproduces the issue, including models and associations

- What do you expect to happen?

- What is actually happening?

- Which dialect you are using (postgres, mysql etc)?

- Which sequelize version you are using?

When you post code, please use [Github flavored markdown](https://help.github.com/articles/github-flavored-markdown), in order to get proper syntax highlighting!

If you can even provide a pull request with a failing unit test, we will love you long time! Plus your issue will likely be fixed much faster.

# Pull requests

We're glad to get pull request if any functionality is missing or something is buggy. However, there are a couple of things you can do to make life easier for the maintainers:

- Explain the issue that your PR is solving - or link to an existing issue

- Make sure that all existing tests pass

- Make sure you followed [coding guidelines](https://github.com/sequelize/sequelize/blob/master/CONTRIBUTING.md#coding-guidelines)

- Add some tests for your new functionality or a test exhibiting the bug you are solving. Ideally all new tests should not pass \_without\_ your changes.

- Use [async/await](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async\_function) in all new tests. Specifically this means:

- don't use `EventEmitter`, `QueryChainer` or the `success`, `done` and `error` events

- don't use a done callback in your test, just return the promise chain.

- Small bugfixes and direct backports to the 4.x branch are accepted without tests.

- If you are adding to / changing the public API, remember to add API docs, in the form of [JSDoc style](http://usejsdoc.org/about-getting-started.html) comments. See [section 4a](#4a-check-the-documentation) for the specifics.

Interested? Coolio! Here is how to get started:

### 1. Prepare your environment

Here comes a little surprise: You need [Node.JS](http://nodejs.org).

### 2. Install the dependencies

Just "cd" into sequelize directory and run `npm ci`, see an example below:

```sh

$ cd path/to/sequelize

$ npm ci

$ npm run tsc

```

### 3. Database

Database instances for testing can be started using Docker or you can use local instances of MySQL and PostgreSQL.

#### 3.a Local instances

For MySQL and PostgreSQL you'll need to create a DB called `sequelize\_test`.

For MySQL this would look like this:

```sh

$ echo "CREATE DATABASE sequelize\_test;" | mysql -uroot

```

\*\*HINT:\*\* by default, your local MySQL install must be with username `root` without password. If you want to customize that, you can set the environment variables `SEQ\_DB`, `SEQ\_USER`, `SEQ\_PW`, `SEQ\_HOST` and `SEQ\_PORT`.

For Postgres, creating the database and (optionally) adding the test user this would look like:

```sh

$ psql

# create database sequelize\_test;

# create user postgres with superuser; -- optional; usually built-in

```

You may need to specify credentials using the environment variables `SEQ\_PG\_USER` and `SEQ\_PG\_PW` when running tests or set a password of 'postgres' for the postgres user on your local database to allow sequelize to connect via TCP to localhost. Refer to `test/config/config.js` for the default credentials and environment variables.

For Postgres you may also need to install the `postgresql-postgis` package (an optional component of some Postgres distributions, e.g. Ubuntu). The package will be named something like: `postgresql-<pg\_version\_number>-postgis-<postgis\_version\_number>`, e.g. `postgresql-9.5-postgis-2.2`. You should be able to find the exact package name on a Debian/Ubuntu system by running the command: `apt-cache search -- -postgis`.

Create the following extensions in the test database:

```

CREATE EXTENSION postgis;

CREATE EXTENSION hstore;

CREATE EXTENSION btree\_gist;

CREATE EXTENSION citext;

```

#### 3.b Docker

Make sure `docker` and `docker-compose` are installed.

If running on macOS, install [Docker for Mac](https://docs.docker.com/docker-for-mac/).

Now launch the docker mysql and postgres servers with this command (you can add `-d` to run them in daemon mode):

```sh

$ docker-compose up postgres-95 mysql-57 mssql

```

> \*\*\_NOTE:\_\*\* If you get the following output:

>

> ```

> ...

> Creating mysql-57 ... error

>

> ERROR: for mysql-57 Cannot create container for service mysql-57: b'create .: volume name is too short, names should be at least two alphanumeric characters'

>

> ERROR: for mysql-57 Cannot create container for service mysql-57: b'create .: volume name is too short, names should be at least two alphanumeric characters'

> ERROR: Encountered errors while bringing up the project.

> ```

>

> You need to set the variables `MARIADB\_ENTRYPOINT` and `MYSQLDB\_ENTRYPOINT` accordingly:

>

> ```sh

> $ export MARIADB\_ENTRYPOINT="$PATH\_TO\_PROJECT/test/config/mariadb"

> $ export MYSQLDB\_ENTRYPOINT="$PATH\_TO\_PROJECT/test/config/mysql"

> ```

\*\*MSSQL:\*\* Please run `npm run setup-mssql` to create the test database.

\*\*POSTGRES:\*\* Sequelize uses [special](https://github.com/sushantdhiman/sequelize-postgres) Docker image for PostgreSQL, which install all the extensions required by tests.

### 4. Running tests

All tests are located in the `test` folder (which contains the

lovely [Mocha](https://mochajs.org/) tests).

```sh

$ npm run test-all || test-mysql || test-sqlite || test-mssql || test-postgres || test-postgres-native

$ # alternatively you can pass database credentials with $variables when testing

$ DIALECT=dialect SEQ\_DB=database SEQ\_USER=user SEQ\_PW=password npm test

```

For docker users you can use these commands instead

```sh

$ DIALECT=mysql npm run test-docker # Or DIALECT=postgres for Postgres SQL

# Only integration tests

$ DIALECT=mysql npm run test-docker-integration

```

### 5. Commit

Sequelize follows the [AngularJS Commit Message Conventions](https://docs.google.com/document/d/1QrDFcIiPjSLDn3EL15IJygNPiHORgU1\_OOAqWjiDU5Y/edit#heading=h.em2hiij8p46d).

Example:

feat(pencil): add 'graphiteWidth' option

Commit messages are used to automatically generate a changelog. They will be validated automatically using [commitlint](https://github.com/marionebl/commitlint)

Then push and send your pull request. Happy hacking and thank you for contributing.

# Coding guidelines

Have a look at our [.eslintrc.json](https://github.com/sequelize/sequelize/blob/master/.eslintrc.json) file for the specifics. As part of the test process, all files will be linted, and your PR will \*\*not\*\* be accepted if it does not pass linting.

# Contributing to the documentation

For contribution guidelines for the documentation, see [CONTRIBUTING.DOCS.md](https://github.com/sequelize/sequelize/blob/master/CONTRIBUTING.DOCS.md).

# Publishing a release (For Maintainers)

1. Ensure that latest build on master is green

2. Ensure your local code is up to date (`git pull origin master`)

3. `npm version patch|minor|major` (see [Semantic Versioning](http://semver.org))

4. Update changelog to match version number, commit changelog

5. `git push --tags origin master`

6. `npm publish .`

7. Copy changelog for version to release notes for version on github