# Contributing to NGXS

:+1::tada: First off, thanks for taking the time to contribute! :tada::+1:

The following is a set of guidelines for contributing to NGXS and its packages.

#### Table Of Contents

[I don't want to read this whole thing, I just have a question!!!](#i-dont-want-to-read-this-whole-thing-i-just-have-a-question)

[How can I help?](#how-can-I-help?)

[Developing](#developing)

[Building](#building)

- [Linking local version](#linking)

[Publish new version](#publishing-new-version)

- [Dev builds](#dev-builds)

- [Beta builds](#beta-builds)

- [Release builds](#release-builds)

[Commit Message Guidelines](#commit)

## I don't want to read this whole thing I just have a question!!!

> \*\*Note:\*\* Please don't file an issue to ask a question.

- Read our [docs](https://ngxs.gitbook.io/ngxs)

- Ask a question in our [slack](https://now-examples-slackin-eqzjxuxoem.now.sh/).

- Ask a question on [stackoverflow](https://stackoverflow.com/questions/tagged/ngxs).

# How can I help?

Check the [issues](https://github.com/ngxs/store/issues) where we have labels for help wanted.

Also chat us up on slack, where we will be more than happy to get you started on an issue.

# Developing

Start by installing all dependencies

```bash

yarn

```

You can then either start develping using TDD

```bash

yarn test

```

Or start the integration app and start playing around

```bash

yarn start

```

## Building

Since our library is a combination of the main `@ngxs/store` and the plugins, we need to build them into separate npm packages and publish them.

running

```bash

yarn build

```

will create a build for each package in the `builds/` directory.

### Linking

If you want to test out a local build in your own app you can create a global symlink for the package which can then be linked locally in your app.

```bash

# build ngxs

yarn build

# cd into the build you want to link

cd builds/store

# create the global symlink

yarn link

```

now navigate to your app then run

```bash

yarn link @ngxs/store

```

now `node\_modules/@ngxs/store` will be symlinked to the local build directory of `@ngxs/store`.

to make a change, save the file in ngxs, run `yarn build` and your app should now be using the new build.

# Publishing new version

> (@ngxs team members only)

## Dev builds

CircleCI picks up all commits to master, which is our development branch. It then runs the build, the tests and the integration tests. If they all succeed,

CircleCI will publish to Code Climate, and to npm using the git commit sha1 as an identified `3.0.0-dev.6e59e7b` and tag it as `@dev`

# Beta builds

\*\*@todo\*\* need to figure out how to get the prerelease version and use it as the git tag, before we can automate the `@beta` releases.

## Release builds

CircleCI picks up when a git tag is created and does the same thing as for the development build, but instead of releasing a dev build,

releases the version that was specified in the root package.json in the git tag. It also tags it as @latest.

# <a name="commit"></a> Commit Message Guidelines

We have very precise rules over how our git commit messages can be formatted. This leads to \*\*more

readable messages\*\* that are easy to follow when looking through the \*\*project history\*\*. But also,

we use the git commit messages to \*\*generate the Angular change log\*\*.

### Commit Message Format

Each commit message consists of a \*\*header\*\*, a \*\*body\*\* and a \*\*footer\*\*. The header has a special

format that includes a \*\*type\*\*, a \*\*scope\*\* and a \*\*subject\*\*:

```

<type>(<scope>): <subject>

<BLANK LINE>

<body>

<BLANK LINE>

<footer>

```

The \*\*header\*\* is mandatory and the \*\*scope\*\* of the header is optional.

Any line of the commit message cannot be longer 100 characters! This allows the message to be easier

to read on GitHub as well as in various git tools.

The footer should contain a [closing reference to an issue](https://help.github.com/articles/closing-issues-via-commit-messages/) if any.

Samples: (even more [samples](https://github.com/ngxs/store/commits/master))

```

docs(changelog): update changelog to beta.5

```

```

fix(release): need to depend on latest rxjs

The version in our package.json gets copied to the one we publish, and users need the latest of these.

```

### Revert

If the commit reverts a previous commit, it should begin with `revert:`, followed by the header of the reverted commit. In the body it should say: `This reverts commit <hash>.`, where the hash is the SHA of the commit being reverted.

### Type

Must be one of the following:

- \*\*build\*\*: Changes that affect the build system or external dependencies (example scopes: gulp, broccoli, npm)

- \*\*ci\*\*: Changes to our CI configuration files and scripts (example scopes: Travis, Circle, BrowserStack, SauceLabs)

- \*\*docs\*\*: Documentation only changes

- \*\*feat\*\*: A new feature

- \*\*fix\*\*: A bug fix

- \*\*perf\*\*: A code change that improves performance

- \*\*refactor\*\*: A code change that neither fixes a bug nor adds a feature

- \*\*style\*\*: Changes that do not affect the meaning of the code (white-space, formatting, missing semi-colons, etc)

- \*\*test\*\*: Adding missing tests or correcting existing tests

### Scope

The scope should be the name of the npm package affected (as perceived by the person reading the changelog generated from commit messages.

The following is the list of supported scopes:

- \*\*devtools-plugin\*\*

- \*\*form-plugin\*\*

- \*\*logger-plugin\*\*

- \*\*router-plugin\*\*

- \*\*storage-plugin\*\*

- \*\*websocket-plugin\*\*

- \*\*store\*\*

- \*\*cli\*\*