# Contributing to NativeScript

:+1: First of all, thank you for taking the time to contribute! :+1:

Here are some guides on how to do that:

- [Code of Conduct](#coc)

- [Reporting Bugs](#bugs)

- [Requesting New Features](#features)

- [Submitting a PR](#pr)

- [Check test report](#test-report)

- [Commit Message Guidelines](#commit-messages)

- [Releasing new versions](#release)

## <a name="coc"></a> Code of Conduct

Help us keep a healthy and open community. We expect all participants in this project to adhere to the [NativeScript Code Of Conduct](https://github.com/NativeScript/codeofconduct).

## <a name="bugs"></a> Reporting Bugs

1. Always update to the most recent master release; the bug may already be resolved.

2. Search for similar issues in the issues list for this repo; it may already be an identified problem.

3. If this is a bug or problem that is clear, simple, and is unlikely to require any discussion -- it is OK to open an issue on GitHub with a reproduction of the bug including workflows and screenshots. If possible, submit a Pull Request with a failing test, entire application or module. If you'd rather take matters into your own hands, fix the bug yourself (jump down to the [Submitting a PR](#pr) section).

> While we are doing all we can to take care of every issue, sometimes we get overwhelmed. That's why

> - issues that are not constructive or describe problems that cannot be reproduced will be closed

> - feature requests or bug reports with unanswered questions regarding the behavior/reproduction for more than 20 days will be closed

## <a name="features"></a> Requesting Features

1. Use Github Issues to submit feature requests.

2. First, search for a similar request and extend it if applicable. This way it would be easier for the community to track the features.

3. When requesting a new feature, please provide as much detail as possible about why you need the feature in your apps. We prefer that you explain a need rather than explain a technical solution for it. That might trigger a nice conversation on finding the best and broadest technical solution to a specific need.

## <a name="pr"></a> Submitting a PR

Before you begin:

\* Read and sign the [NativeScript Contribution License Agreement](http://www.nativescript.org/cla).

\* Make sure there is an issue for the bug or feature you will be working on.

Following these steps is the best way to get you code included in the project:

1. Fork and clone the NativeScript repo:

```bash

git clone https://github.com/<your-git-username>/NativeScript.git

# Navigate to the newly cloned directory

cd NativeScript

# Add an "upstream" remote pointing to the original {N} repo.

git remote add upstream https://github.com/NativeScript/NativeScript.git

```

2. Set up the project (for detailed info check our [development workflow guide](DevelopmentWorkflow.md)):

```bash

#In the repo root

npm install

npm run setup

```

3. Create a branch for your PR

```bash

git checkout -b <my-fix-branch> master

```

4. The fun part! Make your code changes. Make sure you:

- Follow the [code conventions guide](CodingConvention.md).

- Follow the [guide on handling errors and exceptions](HandlingErrors.md).

- Write unit tests for your fix or feature. Check out [writing unit tests guide](WritingUnitTests.md).

5. Before you submit your PR:

- Rebase your changes to the latest master: `git pull --rebase upstream master`.

- Ensure all unit test are green for Android and iOS. Check [running unit tests](DevelopmentWorkflow.md#running-unit-tests).

- Ensure your changes pass tslint validation. (run `npm run tslint` in the root of the repo).

- If you've made changes to a public API, make sure you update and add the `api-reports/NativeScript.api.md` file to the PR. (run `npm run api-extractor` to update the api-report and definitions).

6. Push your fork. If you have rebased you might have to use force-push your branch:

```

git push origin <my-fix-branch> --force

```

7. [Submit your pull request](https://github.com/NativeScript/NativeScript/compare). Please, fill in the Pull Request template - it will help us better understand the PR and increase the chances of it getting merged quickly.

It's our turn from there on! We will review the PR and discuss changes you might have to make before merging it! Thanks!

>Note: Sometimes you will see someone from the contributors team writing strange comments like: `test` or `test branch\_functional\_tests#css-gradients-tests branch\_widgets#vultix/css-gradients` - don't worry about it, these are just phrases that trigger the internal CI builds.

## <a name="test-report"></a> Check test report

The purpose of the test report view is to show the tests' results for the PRs for the external contributors. When a NativeScript team member triggers the tests, you can review the result by selecting the `Details` button next to the `ci/jenkins/core-modules-tests` task.

> Note: the `Details` button will be available when the execution of the test completes and there is at least one failing test.

When you click on the button, you will be redirected to the report page. On the left pane you can find a list of all failed jobs.

> Note: Each item name consists of the application name, type of device and platform version: `pr-e2e-tests-[application-name]-[device-type]-[platform-version]`. Usually, the test applications, that are executed for PRs are part of NativeScript repository.

Based on the executed suite, one of the following or all of the following files will be generated: `mochawesome.html` | `index.html` | `unit-tests.log`. Some of the reports also might include `\*.png`, `\*.logs` or `[page source].xml` files that can help in understanding where is the problem.

For example:

1. When you select the `index.html` page, an additional `TestNG Results` sidebar will be displayed. There you can find a list of all failures.

2. When you select one of them, you will see on the right side all tests, that have been executed. The problematic ones will be coloured in red.

3. If you click on one of them, detailed info or error log will be displayed. As we've mentioned above in some of the test reports, you will also find screenshots, that demonstrates the problem visually. Those images can be found below the info/ error log.

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

Please follow the git commit message format described below when committing your changes or submitting a PR. That allows us to use the commit messages to generate a change log for every new release.

### 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.

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

Samples:

```

docs(README): add link to NativeScript roadmap

```

```

fix(ios-layouts): switch contentInsetAdjustmentBehavior on ScrollView

Do not auto adjust the ScrollView insets, by disabling contentInsetAdjustmentBehavior property, when ScrollView is used as a root Page element.

```

```

release: cut the 3.2.0 release

```

### 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: npm)

\* \*\*ci\*\*: Changes to our CI configuration files and scripts (example scopes: Travis, Jenkins)

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

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

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

\* \*\*fix-next\*\*: A fix for a bug in the master branch, that's not yet released

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

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

\* \*\*release\*\*: Reference commit for the git tag of the release

\* \*\*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 (example scopes: e2e, name-of-the-test-app)

### Scope

The scope should be the name of the affected component in the code.

If the change affects only one of the target platforms (android, ios) that should be specified in the scope.

Sample scopes for the `tns-core-modules` package:

\* \*\*android/application\*\*

\* \*\*ios/application\*\*

\* \*\*action-bar\*\*

\* \*\*animations\*\*

\* \*\*date-picker\*\*

\* \*\*profiling\*\*

### Subject

The subject contains succinct description of the change:

\* use the imperative, present tense: "change" not "changed" nor "changes"

\* don't capitalize first letter

\* no dot (.) at the end

### Body

Just as in the \*\*subject\*\*, use the imperative, present tense: "change" not "changed" nor "changes".

The body should include the motivation for the change and contrast this with previous behavior.

### Footer

The footer should contain any information about \*\*Breaking Changes\*\* and is also the place to

reference GitHub issues that this commit \*\*Closes\*\*.

\*\*Breaking Changes\*\* should start with the word `BREAKING CHANGE:` with a space or two newlines. Describe the changes and provide a migration guide for any code that may be affected.

Sample:

```

feat: Angular 4.2 support

BREAKING CHANGES:

NativeScriptModule should be imported only in the root application

module (usually named AppModule).

ApplicationModule.

All other NgModules in the app (both feature and lazy-loaded ones)

should import the NativeScriptCommonModule instead.

The behavior is aligned with BrowserModule and CommonModule in web

Angular apps. angular.io/guide/ngmodule-faq#q-browser-vs-common-module

Migration steps:

In all NgModules, instead of the root one, replace:

import { NativeScriptModule } from "nativescript-angular/nativescript.module";

@NgModule({

imports: [

NativeScriptModule,

]

})

with:

import { NativeScriptCommonModule } from "nativescript-angular/common";

@NgModule({

imports: [

NativeScriptCommonModule,

]

})

```

The above guidelines are based on the [AngularJS Git Commit Message Conventions][commit-message-format]. A detailed explanation and additional examples can be found in this [document][commit-message-format].

## Where to Start

If you want to contribute, but you are not sure where to start - look for [issues labeled `help wanted`](https://github.com/NativeScript/NativeScript/issues?q=is%3Aopen+is%3Aissue+label%3A%22help+wanted%22).

[commit-message-format]: https://docs.google.com/document/d/1QrDFcIiPjSLDn3EL15IJygNPiHORgU1\_OOAqWjiDU5Y/edit#

## <a name="release"></a> Releasing new versions

Instructions how to release a new version for \*\*NativeScript Core Team Members\*\*.

![](./release-contribution-guide-schema.png?raw=true)

### Here are the steps described in the diagram above.

1. Checkout release branch

```

git checkout release

```

#### If we prepare major or minor release, merge master in release branch else \*\*skip this step\*\*.

```

git merge --ff-only origin/master

```

\*\*\* Note: If there are commits in release branch which are not merged in master branch '-ff-merge' command will fail.

In this case the commits should be merge firstly from release in master branch as explained in section 'Merge changes from release into master' and then repeat step 1.

2. Execute `npm i` to install dependencies:

```

npm i

```

3. Execute [`npm version`](https://docs.npmjs.com/cli/version) to bump the version of `tns-platform-declarations`:

```

cd tns-platform-declarations

npm --no-git-tag-version version [major|minor|patch] -m "release: cut the %s release"

cd ..

```

4. Execute [`npm version`](https://docs.npmjs.com/cli/version) to bump the version of `tns-core-modules`,

tag the release and update the CHANGELOG.md. Don't forget to check the auto-generated CHANGELOG.md

```

cd nativescript-core

npm --no-git-tag-version version [major|minor|patch] -m "release: cut the %s release"

cd ..

```

5. Execute [`npm version`](https://docs.npmjs.com/cli/version) to bump the version of `tns-core-modules`,

tag the release and update the CHANGELOG.md. Don't forget to check the auto-generated CHANGELOG.md

```

cd tns-core-modules-package

npm --no-git-tag-version version [major|minor|patch] -m "release: cut the %s release"

cd ..

```

6. Set correct version of \*\*tns-core-modules-widgets\*\* in tns-core-modules/package.json.

Usually tns-core-modules-widgets should already have been released and we need to set the official version.

7. Create release-branch with change log

```

git checkout -b release-[version]

```

7. Add changes

```

git add changed-files

git commit -m "release: cut the %s release"

git push

```

8. Create git tag as git tag 6.3.0 or use git UI

```

git tag [release-version]

git push --tags

```

9. Create a pull request. Be careful to base your branch on the correct branch

```

curl -d '{"title": "release: cut the [release-version] release","body": "docs: update changelog","head": "${BRANCH}","base": "release"}' -X POST https://api.github.com/repos/NativeScript/NativeScript/pulls -H "Authorization: token ${GIT\_TOKEN}"

```

10. Merge PR into release branch.

11. If all checks has passed publish package. Usually the night builds will be triggered and the package will be ready to be released on the next day.

## Merge changes from release into master

![](./merge-guidance-schema.png)

### Here are steps described in the diagram above.

1. Make sure you are in release branch:

```

git checkout release

git pull

```

2. Create PR to merge changes back in master and preserve history:

```

export MERGE\_BRANCH='merge-release-in-master'

git checkout -b ${MERGE\_BRANCH}

git push --set-upstream origin ${MERGE\_BRANCH}

git merge origin/master

```

3. Resolve conflicts. Choose to keep the version of master branch. If it is needed to revert versions of modules, see at the bottom.

4. Add conflicts:

```

git add resolved files

```

5. Commit changes with default merge message:

```

git commit

git push

```

6. Create pull request. Replace replace env ${MERGE\_BRANCH} with its value

```

curl -d '{"title": "chore: merge release in master","body": "chore: merge release in master","head": "merge-release-in-master","base": "master"}' -X POST https://api.github.com/repos/NativeScript/NativeScript/pulls -H "Authorization: token ${GIT\_TOKEN}"

```

\*\*If needed, revert version of modules and platform declarations to take the one from master:\*\*

```

git checkout origin/master tns-platform-declarations/package.json tns-core-modules/package.json

git commit --amend

git push --force-with-lease

```

This will require to repeat steps from 1 to 4, since we need to keep the branches with the same history