# Contributing

## Questions

If you have questions about implementation details, help or support, then please use our dedicated community forum at [Github Discussions](https://github.com/tannerlinsley/react-table/discussions) \*\*PLEASE NOTE:\*\* If you choose to instead open an issue for your question, your issue will be immediately closed and redirected to the forum.

## Reporting Issues

If you have found what you think is a bug, please [file an issue](https://github.com/tannerlinsley/react-table/issues/new). \*\*PLEASE NOTE:\*\* Issues that are identified as implementation questions or non-issues will be immediately closed and redirected to [Github Discussions](https://github.com/tannerlinsley/react-table/discussions)

## Suggesting new features

If you are here to suggest a feature, first create an issue if it does not already exist. From there, we will discuss use-cases for the feature and then finally discuss how it could be implemented.

## Development

If you have been assigned to fix an issue or develop a new feature, please follow these steps to get started:

- Fork this repository

- Install dependencies by running `$ yarn`

- Link `react-table` locally by running `$ yarn link`

- Auto-build files as you edit by running `$ yarn start`

- Implement your changes and tests to files in the `src/` directory and corresponding test files

- To run examples, follow their individual directions. Usually this is just `$ yarn && yarn start`.

- To run examples using your local build, link to the local `react-table` by running `$ yarn link react-table` from the example's directory

- Document your changes in the appropriate doc page

- Git stage your required chnages and commit (see below commit guidelines)

- Submit PR for review

## Commit message conventions

`react-table` is using [Angular Commit Message Conventions](https://github.com/angular/angular.js/blob/master/DEVELOPERS.md#-git-commit-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\*\*.

### 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 than 100 characters! This allows the message to be easier to read on GitHub as well as in various git tools.

### Type

Must be one of the following:

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

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

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

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

semi-colons, etc)

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

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

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

- \*\*chore\*\*: Changes to the build process or auxiliary tools and libraries such as documentation

generation

### Scope

The scope could be anything specifying place of the commit change. For example `useQuery`, `useMutation` etc...

You can use `\*` when the change affects more than a single scope.

### 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](https://help.github.com/en/github/managing-your-work-on-github/linking-a-pull-request-to-an-issue).

\*\*Breaking Changes\*\* should start with the word `BREAKING CHANGE:` with a space or two newlines. The rest of the commit message is then used for this.

### Example

Here is an example of the release type that will be done based on a commit messages:

| Commit message | Release type |

| ------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------ | -------------------------- |

| `fix(pencil): stop graphite breaking when too much pressure applied` | Patch Release |

| `feat(pencil): add 'graphiteWidth' option` | ~~Minor~~ Feature Release |

| `perf(pencil): remove graphiteWidth option`<br><br>`BREAKING CHANGE: The graphiteWidth option has been removed.`<br>`The default graphite width of 10mm is always used for performance reasons.` | ~~Major~~ Breaking 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.

## Pull requests

Maintainers merge pull requests by squashing all commits and editing the commit message if necessary using the GitHub user interface.

Use an appropriate commit type. Be especially careful with breaking changes.

## Releases

For each new commit added to `master` with `git push` or by merging a pull request or merging from another branch, a github action is triggered and runs the `semantic-release` command to make a release if there are codebase changes since the last release that affect the package functionalities.