# Contributing to Viewer.js

> Based on [Angular's contributing guidelines](https://github.com/angular/angular/blob/master/CONTRIBUTING.md).

We would love for you to contribute to Viewer.js and help make it even better than it is today! As a contributor, here are the guidelines we would like you to follow:

- [Code of Conduct](#code-of-conduct)

- [Question or Problem](#question-or-problem)

- [Issues and Bugs](#issues-and-bugs)

- [Feature Requests](#feature-requests)

- [Submission Guidelines](#submission-guidelines)

- [Coding Rules](#coding-rules)

- [Commit Message Guidelines](#commit-message-guidelines)

## Code of Conduct

Help us keep Viewer.js open and inclusive. Please read and follow our [Code of Conduct](CODE\_OF\_CONDUCT.md).

## Question or Problem

Do not open issues for general support questions as we want to keep GitHub issues for bug reports and feature requests. You've got much better chances of getting your question answered on [Stack Overflow](https://stackoverflow.com/questions/tagged/viewerjs) where the questions should be tagged with tag `viewerjs`.

Stack Overflow is a much better place to ask questions since:

- There are thousands of people willing to help on Stack Overflow.

- Questions and answers stay available for public viewing so your question / answer might help someone else.

- Stack Overflow's voting system assures that the best answers are prominently visible.

To save your and our time, we will systematically close all issues that are requests for general support and redirect people to Stack Overflow.

## Issues and Bugs

If you find a bug in the source code, you can help us by [submitting an issue](#submitting-an-issue) to our [GitHub Repository](https://github.com/fengyuanchen/viewerjs). Even better, you can [submit a Pull Request](#submitting-a-pull-request-pr) with a fix.

## Feature Requests

You can \*request\* a new feature by [submitting an issue](#submitting-an-issue) to our [GitHub Repository](https://github.com/fengyuanchen/viewerjs). If you would like to \*implement\* a new feature, please submit an issue with a proposal for your work first, to be sure that we can use it.

Please consider what kind of change it is:

- For a \*\*Major Feature\*\*, first open an issue and outline your proposal so that it can be discussed. This will also allow us to better coordinate our efforts, prevent duplication of work, and help you to craft the change so that it is successfully accepted into the project.

- \*\*Small Features\*\* can be crafted and directly [submitted as a Pull Request](#submitting-a-pull-request-pr).

## Submission Guidelines

### Submitting an Issue

Before you submit an issue, please search the [issue tracker](https://github.com/fengyuanchen/viewerjs/issues), maybe an issue for your problem already exists and the discussion might inform you of workarounds readily available.

We want to fix all the issues as soon as possible, but before fixing a bug we need to reproduce and confirm it. In order to reproduce bugs we will systematically ask you to provide a minimal reproduction scenario using [CodePen](https://codepen.io/pen). Having a live, reproducible scenario gives us wealth of important information without going back & forth to you with additional questions like:

- version of Viewer.js used

- 3rd-party libraries and their versions

- and most importantly - a use-case that fails

A minimal reproduce scenario using [CodePen](https://codepen.io/pen) allows us to quickly confirm a bug (or point out coding problem) as well as confirm that we are fixing the right problem. If [CodePen](https://codepen.io/pen) is not a suitable way to demonstrate the problem (for example for issues related to our npm packaging), please create a standalone git repository demonstrating the problem.

We will be insisting on a minimal reproduce scenario in order to save maintainers time and ultimately be able to fix more bugs. Interestingly, from our experience users often find coding problems themselves while preparing a minimal reproduce scenario. We understand that sometimes it might be hard to extract essentials bits of code from a larger code-base but we really need to isolate the problem before we can fix it.

Unfortunately we are not able to investigate / fix bugs without a minimal reproduce scenario, so if we don't hear back from you we are going to close an issue that don't have enough info to be reproduced.

You can file new issues by filling out our [new issue form](https://github.com/fengyuanchen/viewerjs/issues/new).

### Submitting a Pull Request (PR)

Before you submit your Pull Request (PR) consider the following guidelines:

1. Search [GitHub](https://github.com/fengyuanchen/viewerjs/pulls) for an open or closed PR that relates to your submission. You don't want to duplicate effort.

1. Fork the \*\*fengyuanchen/viewerjs\*\* repo.

1. Make your changes in a new git branch:

```shell

git checkout -b my-fix-branch master

```

1. Create your patch, \*\*including appropriate test cases\*\*.

1. Follow our [Coding Rules](#coding-rules).

1. Run the full Viewer.js test suite, and ensure that all tests pass.

1. Commit your changes using a descriptive commit message that follows our [Commit Message Guidelines](#commit-message-guidelines). Adherence to these guidelines is necessary because release notes are automatically generated from these messages.

```shell

git commit -a

```

Note: the optional commit `-a` command line option will automatically "add" and "rm" edited files.

1. Push your branch to GitHub:

```shell

git push origin my-fix-branch

```

1. In GitHub, send a pull request to `viewerjs:master`.

1. If we suggest changes then:

- Make the required updates.

- Re-run the Viewer.js test suites to ensure tests are still passing.

- Rebase your branch and force push to your GitHub repository (this will update your Pull Request):

```shell

git rebase master -i

git push -f

```

That's it! Thank you for your contribution!

#### After your pull request is merged

After your pull request is merged, you can safely delete your branch and pull the changes from the main (upstream) repository:

1. Delete the remote branch on GitHub either through the GitHub web UI or your local shell as follows:

```shell

git push origin --delete my-fix-branch

```

1. Check out the master branch:

```shell

git checkout master -f

```

1. Delete the local branch:

```shell

git branch -D my-fix-branch

```

1. Update your master with the latest upstream version:

```shell

git pull --ff upstream master

```

## Coding Rules

To ensure consistency throughout the source code, keep these rules in mind as you are working:

- All features or bug fixes \*\*must be tested\*\* by one or more specs (unit-tests).

- All public API methods \*\*must be documented\*\*.

- We follow [Airbnb's JavaScript Style Guide](https://github.com/airbnb/javascript).

## Commit Message Guidelines

### Commit Message Format

A commit message consists of a \*\*header\*\*, \*\*body\*\* and \*\*footer\*\*. The header has a \*\*type\*\*, \*\*scope\*\* and \*\*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.

Here are some [samples](https://github.com/fengyuanchen/viewerjs/commits/master).

### 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 could be anything specifying place of the commit change. For example `show`, `view`, `play`, etc...

### 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. The rest of the commit message is then used for this.