# Contributing to LCUI

We would love for you to contribute to LCUI 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][coc]

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

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

- [Feature Requests](#feature-requests)

- [Submission Guidelines](#submission-guidelines)

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

## 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][stackoverflow] where the questions should be tagged with tag `lcui`.

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][github]. Even better, you can

[submit a Pull Request](#submitting-a-pull-request) with a fix.

## Feature Requests

You can \*request\* a new feature by [submitting an issue](#submitting-an-issue) to our GitHub

Repository. 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).

## Submission Guidelines

### Submitting an Issue

Before you submit an issue, please search the issue tracker, 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. Having a live, reproducible scenario gives us wealth of important information without going back & forth to you with additional questions like:

- version of LCUI used

- opreation system version

- 3rd-party libraries and their versions

- and most importantly - a use-case that fails

### Submitting a Pull Request

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

- The `master` branch is basically just a snapshot of the latest stable release. All development should be done in dedicated branches. \*\*Do not submit PRs against the `master` branch.\*\*

- Checkout a topic branch from the relevant branch, e.g. `develop`, and merge back against that branch.

- It's OK to have multiple small commits as you work on the PR - we will let GitHub automatically squash it before merging.

- Make sure `make test` passes.

- If adding new feature:

- Add accompanying test case.

- Provide convincing reason to add this feature. Ideally you should open a suggestion issue first and have it greenlighted before working on it.

- If fixing a bug:

- If you are resolving a special issue, add `(fix #xxxx[,#xxx])` (#xxxx is the issue id) in your PR title for a better release log, e.g. `update entities encoding/decoding (fix #3899)`.

- Provide detailed description of the bug in the PR. Live demo preferred.

- Add appropriate test coverage if applicable.

## 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\*\*. (Details TBC).

- Follow the existing code style. You have the following two ways to format the code:

- Use [clang-format](http://clang.llvm.org/docs/ClangFormat.html) to format the changed files: `clang-format --style=file [src/????.c]`.

- Install the [NodeJS](https://nodejs.org/en/) Environment and run `npm install`, which will add a git hook to format the changed code, and it will run before you run `git commit`.

## 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 LCUI 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\*\*:

``` text

<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: (even more [samples](https://github.com/lc-soft/LCUI/commits/master))

``` text

docs(changelog): update change log to beta.1

```

Use this format, we can easily filter out commits that we do not want to see, for example:

``` shell

# show commits that contain new features

git log --pretty=format:"%h %ad %s %d" --date=short --grep "^feat"

# show commits that contain functional code changes (including features, bug fixes, refactorings and improvements)

git log --pretty=format:"%h %ad %s %d" --date=short --grep "^\(feat\|fix\|refactor\|perf\)"

```

### 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

- \*\*ci\*\*: Changes to our CI configuration files and scripts

- \*\*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 `timer`, `mainloop`, `renderer`, `layout`, `css`, `font`, `widget` 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.

[coc]: CODE\_OF\_CONDUCT.md

[github]: https://github.com/lc-soft/LCUI

[stackoverflow]: http://stackoverflow.com/questions/tagged/LCUI