# Vue.js Contributing Guide

Hi! I'm really excited that you are interested in contributing to Vue.js. Before submitting your contribution, please make sure to take a moment and read through the following guidelines:

- [Code of Conduct](https://github.com/vuejs/vue/blob/dev/.github/CODE\_OF\_CONDUCT.md)

- [Issue Reporting Guidelines](#issue-reporting-guidelines)

- [Pull Request Guidelines](#pull-request-guidelines)

- [Development Setup](#development-setup)

- [Project Structure](#project-structure)

- [Contributing Tests](#contributing-tests)

- [Financial Contribution](#financial-contribution)

## Issue Reporting Guidelines

- Always use [https://new-issue.vuejs.org/](https://new-issue.vuejs.org/) to create new issues.

## Pull Request Guidelines

- Checkout a topic branch from a base branch, e.g. `master`, and merge back against that branch.

- If adding a new feature:

- Add accompanying test case.

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

- If fixing bug:

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

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

- Add appropriate test coverage if applicable. You can check the coverage of your code addition by running `yarn test --coverage`.

- It's OK to have multiple small commits as you work on the PR - GitHub can automatically squash them before merging.

- Make sure tests pass!

- Commit messages must follow the [commit message convention](./commit-convention.md) so that changelogs can be automatically generated. Commit messages are automatically validated before commit (by invoking [Git Hooks](https://git-scm.com/docs/githooks) via [yorkie](https://github.com/yyx990803/yorkie)).

- No need to worry about code style as long as you have installed the dev dependencies - modified files are automatically formatted with Prettier on commit (by invoking [Git Hooks](https://git-scm.com/docs/githooks) via [yorkie](https://github.com/yyx990803/yorkie)).

## Development Setup

You will need [Node.js](http://nodejs.org) \*\*version 10+\*\*, and [Yarn 1.x](https://yarnpkg.com/en/docs/install).

After cloning the repo, run:

```bash

$ yarn # install the dependencies of the project

```

A high level overview of tools used:

- [TypeScript](https://www.typescriptlang.org/) as the development language

- [Rollup](https://rollupjs.org) for bundling

- [Jest](https://jestjs.io/) for unit testing

- [Prettier](https://prettier.io/) for code formatting

## Scripts

### `yarn build`

The `build` script builds all public packages (packages without `private: true` in their `package.json`).

Packages to build can be specified with fuzzy matching:

```bash

# build runtime-core only

yarn build runtime-core

# build all packages matching "runtime"

yarn build runtime --all

```

#### Build Formats

By default, each package will be built in multiple distribution formats as specified in the `buildOptions.formats` field in its `package.json`. These can be overwritten via the `-f` flag. The following formats are supported:

- \*\*`global`\*\*

- \*\*`esm-bundler`\*\*

- \*\*`esm-browser`\*\*

- \*\*`cjs`\*\*

Additional formats that only apply to the main `vue` package:

- \*\*`global-runtime`\*\*

- \*\*`esm-bundler-runtime`\*\*

- \*\*`esm-browser-runtime`\*\*

More details about each of these formats can be found in the [`vue` package README](https://github.com/vuejs/vue-next/blob/master/packages/vue/README.md#which-dist-file-to-use) and the [Rollup config file](https://github.com/vuejs/vue-next/blob/master/rollup.config.js).

For example, to build `runtime-core` with the global build only:

```bash

yarn build runtime-core -f global

```

Multiple formats can be specified as a comma-separated list:

```bash

yarn build runtime-core -f esm-browser,cjs

```

#### Build with Source Maps

Use the `--sourcemap` or `-s` flag to build with source maps. Note this will make the build much slower.

#### Build with Type Declarations

The `--types` or `-t` flag will generate type declarations during the build and in addition:

- Roll the declarations into a single `.d.ts` file for each package;

- Generate an API report in `<projectRoot>/temp/<packageName>.api.md`. This report contains potential warnings emitted by [api-extractor](https://api-extractor.com/).

- Generate an API model json in `<projectRoot>/temp/<packageName>.api.json`. This file can be used to generate a Markdown version of the exported APIs.

### `yarn dev`

The `dev` script bundles a target package (default: `vue`) in a specified format (default: `global`) in dev mode and watches for changes. This is useful when you want to load up a build in an HTML page for quick debugging:

```bash

$ yarn dev

> rollup v1.19.4

> bundles packages/vue/src/index.ts ? packages/vue/dist/vue.global.js...

```

- The `dev` script also supports fuzzy match for the target package, but will only match the first package matched.

- The `dev` script supports specifying build format via the `-f` flag just like the `build` script.

- The `dev` script also supports the `-s` flag for generating source maps, but it will make rebuilds slower.

### `yarn dev-compiler`

The `dev-compiler` script builds, watches and serves the [Template Explorer](https://github.com/vuejs/vue-next/tree/master/packages/template-explorer) at `http://localhost:5000`. This is extremely useful when working on the compiler.

### `yarn test`

The `test` script simply calls the `jest` binary, so all [Jest CLI Options](https://jestjs.io/docs/en/cli) can be used. Some examples:

```bash

# run all tests

$ yarn test

# run tests in watch mode

$ yarn test --watch

# run all tests under the runtime-core package

$ yarn test runtime-core

# run tests in a specific file

$ yarn test fileName

# run a specific test in a specific file

$ yarn test fileName -t 'test name'

```

## Project Structure

This repository employs a [monorepo](https://en.wikipedia.org/wiki/Monorepo) setup which hosts a number of associated packages under the `packages` directory:

- `reactivity`: The reactivity system. It can be used standalone as a framework-agnostic package.

- `runtime-core`: The platform-agnostic runtime core. Includes code for the virtual dom renderer, component implementation and JavaScript APIs. Higher-order runtimes (i.e. custom renderers) targeting specific platforms can be created using this package.

- `runtime-dom`: The runtime targeting the browser. Includes handling of native DOM API, attributes, properties, event handlers etc.

- `runtime-test`: The lightweight runtime for testing. Can be used in any JavaScript environment since it "renders" a tree of plain JavaScript objects. The tree can be used to assert correct render output. Also provides utilities for serializing the tree, triggering events, and recording actual node operations performed during an update.

- `server-renderer`: Package for server-side rendering.

- `compiler-core`: The platform-agnostic compiler core. Includes the extensible base of the compiler and all platform-agnostic plugins.

- `compiler-dom`: Compiler with additional plugins specifically targeting the browser.

- `compiler-ssr`: Compiler that produces render functions optimized for server-side rendering.

- `template-explorer`: A development tool for debugging compiler output. You can run `yarn dev template-explorer` and open its `index.html` to get a repl of template compilation based on current source code.

A [live version](https://vue-next-template-explorer.netlify.com) of the template explorer is also available, which can be used for providing reproductions for compiler bugs. You can also pick the deployment for a specific commit from the [deploy logs](https://app.netlify.com/sites/vue-next-template-explorer/deploys).

- `shared`: Internal utilities shared across multiple packages (especially environment-agnostic utils used by both runtime and compiler packages).

- `vue`: The public facing "full build" which includes both the runtime AND the compiler.

### Importing Packages

The packages can import each other directly using their package names. Note that when importing a package, the name listed in its `package.json` should be used. Most of the time the `@vue/` prefix is needed:

```js

import { h } from '@vue/runtime-core'

```

This is made possible via several configurations:

- For TypeScript, `compilerOptions.path` in `tsconfig.json`

- For Jest, `moduleNameMapper` in `jest.config.js`

- For plain Node.js, they are linked using [Yarn Workspaces](https://yarnpkg.com/blog/2017/08/02/introducing-workspaces/).

### Package Dependencies

```

+---------------------+

| |

| @vue/compiler-sfc |

| |

+-----+--------+------+

| |

v v

+---------------------+ +----------------------+

| | | |

+------------>| @vue/compiler-dom +--->| @vue/compiler-core |

| | | | |

+----+----+ +---------------------+ +----------------------+

| |

| vue |

| |

+----+----+ +---------------------+ +----------------------+ +-------------------+

| | | | | | |

+------------>| @vue/runtime-dom +--->| @vue/runtime-core +--->| @vue/reactivity |

| | | | | |

+---------------------+ +----------------------+ +-------------------+

```

There are some rules to follow when importing across package boundaries:

- Never use direct relative paths when importing items from another package - export it in the source package and import it at the package level.

- Compiler packages should not import items from the runtime, and vice versa. If something needs to be shared between the compiler-side and runtime-side, it should be extracted into `@vue/shared` instead.

- If a package (A) has a non-type import, or re-exports a type from another package (B), then (B) should be listed as a dependency in (A)'s `package.json`. This is because the packages are externalized in the ESM-bundler/CJS builds and type declaration files, so the dependency packages must be actually installed as a dependency when consumed from package registries.

## Contributing Tests

Unit tests are collocated with the code being tested in each package, inside directories named `\_\_tests\_\_`. Consult the [Jest docs](https://jestjs.io/docs/en/using-matchers) and existing test cases for how to write new test specs. Here are some additional guidelines:

- Use the minimal API needed for a test case. For example, if a test can be written without involving the reactivity system or a component, it should be written so. This limits the test's exposure to changes in unrelated parts and makes it more stable.

- If testing platform agnostic behavior or asserting low-level virtual DOM operations, use `@vue/runtime-test`.

- Only use platform-specific runtimes if the test is asserting platform-specific behavior.

Test coverage is continuously deployed at https://vue-next-coverage.netlify.app/. PRs that improve test coverage are welcome, but in general the test coverage should be used as a guidance for finding API use cases that are not covered by tests. We don't recommend adding tests that only improve coverage but not actually test a meaning use case.

### Testing Type Definition Correctness

This project uses [tsd](https://github.com/SamVerschueren/tsd) to test the built definition files (`\*.d.ts`).

Type tests are located in the `test-dts` directory. To run the dts tests, run `yarn test-dts`. Note that the type test requires all relevant `\*.d.ts` files to be built first (and the script does it for you). Once the `d.ts` files are built and up-to-date, the tests can be re-run by simply running `yarn test-dts`.

## Financial Contribution

As a pure community-driven project without major corporate backing, we also welcome financial contributions via Patreon and OpenCollective.

- [Become a backer or sponsor on Patreon](https://www.patreon.com/evanyou)

- [Become a backer or sponsor on OpenCollective](https://opencollective.com/vuejs)

### What's the difference between Patreon and OpenCollective funding?

Funds donated via Patreon go directly to support Evan You's full-time work on Vue.js. Funds donated via OpenCollective are managed with transparent expenses and will be used for compensating work and expenses for core team members or sponsoring community events. Your name/logo will receive proper recognition and exposure by donating on either platform.

## Credits

Thank you to all the people who have already contributed to Vue.js!

<a href="https://github.com/vuejs/vue/graphs/contributors"><img src="https://opencollective.com/vuejs/contributors.svg?width=890" /></a>