Welcome to the Vega community. Everyone is welcome to contribute. We value all forms of contributions including code reviews, patches, examples, community participation, tutorial, and blog posts. In this document, we outline the guidelines for contributing to the various aspects of the project.

# Contributing

If you find a bug in the code or a mistake in the [documentation](https://vega.github.io/vega-lite/docs/) or want a new feature, you can help us by creating an issue to [our repository](https://github.com/vega/vega-lite), or even submit a pull request (PR).

- For small fixes, please feel free to submit a pull request. Don't worry about creating an issue first.

- For major changes, please discuss it with the community via a GitHub issue first. This will help us coordinate our efforts, prevent duplication of work, and help you to craft the change so that it is successfully accepted into the project.

- One way to use GitHub for this purpose is to submit a [draft pull request](https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests#draft-pull-requests).

- Generally, we name a branch using this pattern `<your 2-3 letters initial>/<topic>`. For example, @kanitw's branch regarding scale type might be called `kw/scale-type`.

See our [issue](.github/ISSUE\_TEMPLATE.md) and [pull request](.github/PULL\_REQUEST\_TEMPLATE.md) templates for more information.

### Looking for a Task to Contribute

You can find [tasks with the "Good first issue" label in the issue tracker :pray:](https://github.com/vega/vega-lite/labels/Good%20first%20issue%20%3Ababy%3A). Please add a comment in issues if you are planning to work on a major task.

### Help Create New Examples

To submit a new example, fork our [example Block](https://bl.ocks.org/domoritz/455e1c7872c4b38a58b90df0c3d7b1b9) and send us a [pull request to add a link](https://github.com/vega/vega-lite/edit/master/site/examples/index.md) to it to our [example gallery](https://vega.github.io/vega-lite/examples/).

## Documentation and Website

The website is under `site/` and the documentation is under `site/docs/`. We use GitHub Pages to publish our documentation when we release a new version. To contribute changes to the documentation or website, simply submit a pull request that changes the corresponding markdown files in `site/`.

Since we only publish the GitHub Pages when we release a new version, it might be slightly outdated compared to `master`. For development, once you have [setup the repository](#repository-setup), you can run `yarn site` to serve the GitHub page locally at [http://localhost:4000/vega-lite/](http://localhost:4000/vega-lite/).

Note that when you checkout different branches, the compiled JavaScript for the website might be reset. You might have to run `yarn build:site` to recompile the JavaScript so that interactive examples work.

### Documentation Guide

General Guides for Markdown Files:

- Wrap properties (`key`) with back ticks.

- Wrap values with back ticks for numbers and booleans (e.g., `5`, `true`) and wrap string values with both back ticks and double quotes (`"red"`).

#### Property Table

To generate property tables from the JSON schema (which is in turn generated from the Typescript interfaces, you can include the `table.html` template. For example, to generate a table that includes `rangeStep`, `scheme`, and `padding` from `Scale`, you can use

```

{% include table.html props="rangeStep,scheme,padding" source="Scale" %}

```

To define a link for types in the table, you can edit `\_data/link.yml`.

For JSDocs comment in the interfaces, please add `\_\_Default value:\_\_` line at the end to describe the property's value.

#### Examples

To include an example specification in the documentation, the specification's `.vl.json` file must be in `examples/specs`. Then you can use the following span tag to include the specification (e.g., for `point\_1d.vl.json`):

```

<span class="vl-example" data-name="point\_1d"></span>

```

Before adding a new example, you might want to search inside `examples/` folder to see if there are other redundant examples that you can reuse.

To name the example file:

- Please begin with mark type and follow by some description for a static single view example. For stacked marks, add `stacked\_` prefix.

- For composite views, please begin the file with the operator name (e.g., `layer`).

- For interactive example, begin with either `interactive\_` or `selection\_`.

- For examples that are only for regression test, begin with `test\_`.

After you push a new branch to GitHub, the CI will automatically run `yarn build:examples` to recompile all examples and push the changed Vega specs and SVG files in `examples/compiled` , so that your branch includes these changes. When you add a new example or update the code, you may run `yarn build:examples` or `yarn build:example <examplename>` (e.g., `yarn build:example bar\_1d`) to see the change locally. However, do \*\*not\*\* include these changes in your commit as different systems produce slightly different SVGs (mainly due to floating point differences). To avoid unnecessary SVG diffs, we should just let the CI generate the images. You're still encouraged to run `yarn build:examples` to make sure that your code does not cause unnecessary changes.

\*\*Notes:\*\*

1. `yarn build:examples` only re-compile SVGs if the output Vega file changes (so it runs way faster). If you want to enforce re-compilation of all SVGs, use `yarn build:examples-full`.

2. To make Travis run `yarn build:examples-full`, include `[SVG]` in your commit message of the last commit in your branch.

3. To run `yarn build:examples`, you need to install [gnu parallel](https://www.gnu.org/software/parallel/). (For Mac, you can simply do `brew install parallel`.)

# Development Guide

## Repository Setup

1. Make sure you have [node.js](https://nodejs.org/en/). For mac users, we recommend using [homebrew](http://brew.sh) and simply run:

```sh

brew install node

```

2. Clone this repository and cd into your local clone of the repository, and install all the npm dependencies. We use [yarn](https://yarnpkg.com/) to have reproducible dependencies:

```sh

git clone https://github.com/vega/vega-lite.git

cd vega-lite

yarn

```

Now you should be able to build and test the code.

3. To serve the website and documentation, you will need [ruby](https://www.ruby-lang.org/en/), [bundler](http://bundler.io/) and [Jekyll](https://help.github.com/articles/using-jekyll-as-a-static-site-generator-with-github-pages/).

For ruby, Mac users can use [homebrew](http://brew.sh) to add it:

```sh

brew install ruby

```

For bundler:

```sh

gem install bundler

```

Or to install the same version that was used to create the bundle:

```sh

gem install bundler -v "$(grep -A 1 "BUNDLED WITH" Gemfile.lock | tail -n 1)"

```

For Jekyll and its dependencies, because we already have the `Gemfile` in the repo, you can simply run:

```sh

pushd site && bundle install && popd

```

## Directory Structure

- `bin/` – Scripts for using Vega-Lite with command line.

- `site/` – Vega-Lite website including documentation.

- `\_data/` – Jekyll data.

- `\_includes/` – Jekyll includes.

- `\_layouts/` – Jekyll layout files.

- `data/` – Example data.

- `examples/` – Example images, specifications, and pages for the website.

- `static/` – Static files for the website.

- `examples/` – Example Vega-Lite specifications.

- `specs` Vega-Lite examples.

- `compiled` The generated Vega specifications and SVG files of the Vega-Lite examples.

- `scripts/` - Scripts for NPM commands.

- `src/` - Main source code directory.

- All interfaces for Vega-Lite syntax should be declared at the top-level of the `src/` folder.

- `src/index.ts` is the root file for Vega-Lite codebase that exports the global `vl` object.

- Other files under `src/` reflect namespace structure. All methods for `vegaLite.xxx` will be in either `src/xxx.ts` or `src/xxx/xxx.ts`. For example, `vegaLite.channel.\*` methods are in `src/channel.ts` while `vegaLite.compile` is in `src/compile/compile.ts`.

- `test/` - Code for unit testing. `test`'s structure reflects `src`'s directory structure. For example, `test/compile/` tests files inside `src/compile/`.

- `test-runtime/` - Code for runtime tests. You can debug the tests by [running puppeteer in debug mode](https://github.com/smooth-code/jest-puppeteer#put-in-debug-mode).

- `typings/` - TypeScript typing declaration for dependencies.

## Understanding How Vega-Lite Works

- The main compiler code is in `src/compile/compile.ts`. To try to understand how Vega-Lite works, first start by reading the `compile` method in the file and try to understand different phases in the compilation process. You can [browse the code online with Sourcegraph](https://sourcegraph.com/github.com/vega/vega-lite/-/blob/src/compile/compile.ts).

## Commands

This section lists commands that are commonly used during development. See `package.json` for other commands.

### Build

You can run `yarn build` to compile Vega-Lite and regenerate `vega-lite-schema.json`.

### Basic Lint & Test & Test Coverage

`yarn test` run linting and all unit-tests respectively. `yarn format` automatically fixes linting issues if possible. `yarn test:inspect` to inspect tests

`yarn test` includes test coverage and generates a report inside `coverage/index.html`. You can see if specific lines are covered in the unit test by running `open coverage/index.html` and browsing through the report.

### Watch tasks

During development, it can be convenient to rebuild automatically or to run tests in the background. You can use:

- `yarn watch:test` to start a watcher task that \*\*lints and runs tests\*\* when any `.ts` file changes.

- `yarn watch:build` to start a watcher task that \*\*re-compiles Vega-Lite\*\* when `.ts` files related to VL change.

### Website

`yarn site`. See details in [Documentation and Website](#documentation-and-website).

### Deployment

To make a new release, bump the version number with `scripts/bump.sh 4.0.0` (you can get the current version with `scripts/bump.sh`). Then push the tagged commit that the script creates, make a new GitHub release for the tag, and update the changelog. `yarn changelog` can help generate initial draft for a changelog, though we should manually add screenshots for feature releases (major and minor). After creating a GitHub release, GitHub will run checks and make a release. This will also update the website and schema. If you want to update only GitHub Pages, use `yarn deploy:site`.

## Suggested Programming Environment.

We use the [Visual Studio Code](https://code.visualstudio.com/) editor.

- VSCode has nice built-in Typescript support!

- We already include project settings to hide compiled files (`\*.js`, `\*.js.map`). This should work automatically if you open the `vega-lite` folder with VSCode.

- Make sure to install [ESLint](https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint), [Prettier](https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode) extensions.

- The [vscode-jest-runner](https://marketplace.visualstudio.com/items?itemName=firsttris.vscode-jest-runner) extension is also very helpful for debugging tests.

## Manually Testing with Vega-Editor

To manually test your changes locally, you should have a local instance of [Vega Editor](https://github.com/vega/editor) and link Vega-Lite to the editor (See [Vega Editor's README](https://github.com/vega/editor#local-testing--debugging) for instructions).

To update the Vega-Lite code in the editor, you need to compile TypeScript to JavaScript. The easiest way is to run `yarn tsc:src -w` in the Vega-Lite directory. This command will automatically recompile the code whenever you make changes.

## Pull Requests and Continuous Integration (CI)

All pull requests will be tested on [GitHub Actions](https://github.com/features/actions). If your PR does not pass the checks, your PR will not be approved. The CI will run `yarn test`, generate Vega specs and SVG files from your updated code, compare them with the existing compiled outputs in `examples/compiled/`, and check code coverage of your code. If you don't want your PR reviewed until checks pass, mark the [pull request as draft](https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/about-pull-requests#draft-pull-requests). Once you're ready for review, convert the pull request to mark it as ready for review.

### Code Coverage

When checking for code coverage, we require that your PR tests cover at least the same percentage of code that was being covered before. To check the code coverage, you can see the link in the job log of your CI test, from the GitHub page of your PR, or on `https://codecov.io/gh/vega/vega-lite/commits`. It'll be usually in the form of `https://codecov.io/gh/vega/vega-lite/commit/your-full-head-commit-number`. Under the \_Files\_ and \_Diff\_ tab, you can check your code coverage differences and total. In \_Files\_, you can check which lines in your files are being tested (marked in green) and which are not (marked in red). We appreciate PRs that improve our overall code coverage!