## 0x Contribution Guide

We welcome contributions from anyone on the internet and are grateful for even the smallest contributions. This document will help get you setup to start contributing back to 0x.

### Getting started

1. Fork `0xproject/0x-monorepo`

2. Clone your fork

3. Follow the [installation & build steps](https://github.com/0xProject/0x-monorepo#install-dependencies) in the repo's top-level README.

4. Setup the recommended [Development Tooling](#development-tooling).

5. Open a PR with the `[WIP]` flag against the `development` branch and describe the change you are intending to undertake in the PR description. (see [our branch naming conventions](#branch-structure))

Before removing the `[WIP]` tag and submitting the PR for review, make sure:

- It passes our linter checks (`yarn lint`)

- It is properly formatted with Prettier (`yarn prettier`)

- It passes our continuous integration tests (See: [Enabling code coverage checks on your fork](#enabling-code-coverage-checks-on-your-fork) for instructions on getting the `submit-coverage` test to pass on forks)

- You've created/updated the corresponding [CHANGELOG](#CHANGELOGs) entries.

- Your changes have sufficient test coverage (e.g regression tests have been added for bug fixes)

### Branch structure

We have two main branches:

- `master` represents the most recently released (published on npm) version of the codebase.

- `development` represents the current development state of the codebase.

ALL PRs should be opened against `development`.

Branch names should be prefixed with `fix`, `feature` or `refactor`.

- e.g `fix/missing-import`

- If the PR only edits a single package, add it's name too

- e.g `fix/subproviders/missing-import`

### CHANGELOGs

At 0x we use [Semantic Versioning](http://semver.org/) for all our published packages. If a change you make corresponds to a semver bump, you must modify the package's `CHANGELOG.json` file accordingly.

Each CHANGELOG entry that corresponds to a published package will have a `timestamp`. If no entry exists without a `timestamp`, you must first create a new one:

```

{

"version": "1.0.1", <- The updated package version

"changes": [

{

"note": "", <- Describe your change

"PR": 100 <- Your PR number

}

]

},

```

If an entry without a `timestamp` already exists, this means other changes have been introduced by other collaborators since the last publish. Add your changes to the list of notes and adjust the version if your PR introduces a greater semver change (i.e current changes required a patch bump, but your changes require a major version bump).

### Development Tooling

We strongly recommend you use the [VSCode](https://code.visualstudio.com/) text editor since most of our code is written in TypeScript and it offers amazing support for the language.

#### Linter

We use [TSLint](https://palantir.github.io/tslint/) with [custom configs](https://github.com/0xProject/0x-monorepo/tree/development/packages/tslint-config) to keep our code-style consistent.

Use `yarn:lint` to lint the entire monorepo, and `PKG={PACKAGE\_NAME} yarn lint` to lint a specific package.

If you want to change a rule, or add a custom rule, please make these changes to our [tslint-config](https://github.com/0xProject/0x-monorepo/tree/development/packages/tslint-config) package. All other packages have it as a dependency.

Integrate it into your text editor:

- VSCode: [vscode-tslint](https://marketplace.visualstudio.com/items?itemName=eg2.tslint)

- Atom: [linter-tslint](https://atom.io/packages/linter-tslint)

#### Auto-formatter

We use [Prettier](https://prettier.io/) to auto-format our code. Be sure to either add a [text editor integration](https://prettier.io/docs/en/editors.html) or a [pre-commit hook](https://prettier.io/docs/en/precommit.html) to properly format your code changes.

If using the Atom text editor, we recommend you install the following packages:

- VSCode: [prettier-vscode](https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode)

- Atom: [prettier-atom](https://atom.io/packages/prettier-atom)

## Unenforced coding conventions

A few of our coding conventions are not yet enforced by the linter/auto-formatter. Be careful to follow these conventions in your PR's.

1. Unused anonymous function parameters should be named with an underscore + number (e.g \\_1, \\_2, etc...)

1. There should be a new-line between methods in a class and between test cases.

1. If a string literal has the same value in two or more places, it should be a single constant referenced in both places.

1. Do not import from a project's `index.ts` (e.g import { Token } from '../src';). Always import from the source file itself.

1. Generic error variables should be named `err` instead of `e` or `error`.

1. If you \_must\_ cast a variable to any - try to type it back as fast as possible. (e.g., `const cw = ((zeroEx as any).\_contractWrappers as ContractWrappers);`). This ensures subsequent code is type-safe.

1. Our enum conventions coincide with the recommended TypeScript conventions, using capitalized keys, and all-caps snake-case values. Eg `GetStats = 'GET\_STATS'`

1. All public, exported methods/functions/classes must have associated Javadoc-style comments.

### Fix `submit-coverage` CI failure

If you simply fork the repo and then create a PR from it, your PR will fail the `submit-coverage` check on CI. This is because the 0x CircleCI configuration sets the `COVERALLS\_REPO\_TOKEN` environment variable to the token for `0xProject/0x-monorepo`, but when running the check against your fork the token needs to match your repo's name `your-username/0x-monorepo`.

To facilitate this check, after creating your fork, but before creating the branch for your PR, do the following:

1. Log in to [coveralls.io](https://coveralls.io/), go to `Add Repos`, and enable your fork. Then go to the settings for that repo, and copy the `Repo Token` identifier.

2. Log in to [CircleCI](https://circleci.com/login), go to `Add Projects`, click the `Set Up Project` button corresponding to your fork, and then click `Start Building`. (Aside from step 3 below, no actual set up is needed, since it will use the `.circleci/config.yml` file in 0x-monorepo, so you can ignore all of the instruction/explanation given on the page with the `Start Building` button.)

3. In CircleCI, configure your project to add an environment variable, with name `COVERALLS\_REPO\_TOKEN`, and for the value paste in the `Repo Token` you copied in step 1.

Now, when you push to your branch, CircleCI will automatically run all of the checks in your own instance, and the coverage check will work since it has the proper `Repo Token`, and the PR will magically refer to your own checks rather than running them in the 0x CircleCI instance.