# Contribution guidelines

Firstly, thanks for taking an interest in these projects!

There're many ways you can contribute, help to improve these projects

and spread the word. This doc describes some loose guidelines for some

of them.

<!-- TOC depthFrom:2 depthTo:3 -->

- [Usage questions & issues](#usage-questions--issues)

- [Join our Discord](#join-our-discord)

- [Submit an issue](#submit-an-issue)

- [Documentation, examples & advocacy](#documentation-examples--advocacy)

- [Changes to readme files](#changes-to-readme-files)

- [Wiki additions, blog posts, examples](#wiki-additions-blog-posts-examples)

- [Small, standalone examples in the repo](#small-standalone-examples-in-the-repo)

- [Contributing code](#contributing-code)

- [Fork it](#fork-it)

- [Create a feature branch](#create-a-feature-branch)

- [Commit your changes](#commit-your-changes)

- [Run & add tests](#run--add-tests)

- [Push to your feature branch](#push-to-your-feature-branch)

- [Create new Pull Request](#create-new-pull-request)

- [Code style guide](#code-style-guide)

- [Project layout](#project-layout)

- [Imports](#imports)

- [Sorted imports](#sorted-imports)

- [No default exports](#no-default-exports)

- [No 3rd party dependencies](#no-3rd-party-dependencies)

- [Intra-repo dependencies](#intra-repo-dependencies)

- [No exported `const enums`](#no-exported-const-enums)

- [Arrow functions preferred](#arrow-functions-preferred)

- [Function / ctor arguments](#function--ctor-arguments)

- [Naming conventions](#naming-conventions)

- [Doc strings](#doc-strings)

- [Formatting](#formatting)

- [Donations](#donations)

<!-- /TOC -->

## Usage questions & issues

### Join our Discord

We have a [Discord community](https://discord.gg/JhYcmBw) where it's

usually the fastest to get some answers to any burning questions and

where we generally share & discuss feedback/ideas/interesting things

loosely related to this project. Come and say Hi!

### Submit an issue

In many cases, it might be better to [submit an

issue](https://github.com/thi-ng/umbrella/issues) here on GitHub,

especially if you've discovered some kind of bug and/or want to propose

new features etc., which require longer discussion. I'd also encourage

using the issue tracker for any topic which might be beneficial to

know/discover for other users in the future (much like a forum), since

this format & discussion has more longevity/discoverability than a

Discord channel.

When submitting an issue, please follow the instructions in the "new

issue" template.

When running into code troubles, please also try to include a minimal

reproducible example. If the example is too large for including in the

issue, please create a gist or repo and add a link to it.

## Documentation, examples & advocacy

Due to the wide scope of this project, documentation is still sparse in

some areas, so any contributions in the form listed below are very

welcome. I primarily can only work on this project in my spare time, so

have to balance between adding new features (always my priority) &

documentation whenever I can.

Please also see further information about [doc strings in the source

code](#doc-strings) below.

### Changes to readme files

The readme files for all packages are generated from their respective

templated versions (`tpl.readme.md` files). Please only ever edit the

template and then re-generate the actual readme like so:

```bash

cd umbrella

lerna run doc:readme --scope @thi.ng/<package-name>

```

### Wiki additions, blog posts, examples

The wiki of this repo is still pretty barebones, however adding some

gallery, FAQs, tutorials, best practice tips, feature ideas/roadmaps and

a collection of links to blog posts, 3rd party examples, etc. is

planned. If you have anything to contribute (or have already done so

elsewhere), please do get in touch.

### Small, standalone examples in the repo

"Learning by example" has been my life's motto. The ~85 examples

included in the repo are each focused on specific features and kept

quite lowkey for reasons of simplicity. Since many of the projects are

meant to be integrated with other 3rd party projects, I'm always on the

lookout for similar small demos showcasing these integrations (e.g. w/

React, WebGL, SVG, etc.)

The repo contains a generator bash script to create a new example

project skeleton in the repo's `/examples` dir. Use it like below (and

make sure the name isn't already taken :)

```bash

cd umbrella

scripts/make-example my-example

cd examples/my-example

```

Currently, it's recommended to install

[Parcel](https://github.com/parcel-bundler/parcel/) globally for running

the example. Please consult the [example build

instructions](https://github.com/thi-ng/umbrella/wiki/Example-build-instructions)

in the wiki.

FWIW I usually launch examples from the repo root via a subshell:

```bash

(cd examples/webgl-msdf && yarn start)

```

If you choose to add a new example this way (and not elsewhere on the

interweb), please also read the next section...

## Contributing code

\*\*IMPORTANT:\*\* To avoid misunderstandings or disappointment, please

always first submit an issue discussing any new feature or large

refactoring before starting to code and submitting PRs. For small bug

fixes or new examples, it's usually fine without, though. I'm not trying

to complicate things, but it's always a good idea to first talk about

larger contributions and there're also various (sometimes still

unpublished) feature branches, incl. some existing WIP relevant to your

issue/proposal...

### Fork it

```bash

git clone https://github.com/thi-ng/umbrella.git

cd umbrella

# Installs all dev deps & builds all packages

yarn build

```

### Create a feature branch

Creating feature branches is only needed for adding new code or larger

fixes/refactoring. If you're only updating some docs or other minor

fixes you can work on your own `develop` branch directly...

\*\*Always use `develop` as base branch, which also is the default branch

of this repo...\*\*

```bash

git checkout -b feature/my-feature develop

```

This repo is using the `git-flow` branching model and all new

development should be done on feature branches based off the current

`develop` branch. PR's submitted directly against the `master` branch

WILL be refused (with a few exceptions).

With the `git-flow` CLI tool installed, you can also run:

```bash

git-flow feature start my-feature

```

### Commit your changes

```bash

git commit -am 'feat(module): description'

```

\*\*Please do use the [Conventional Commits

convention](https://conventionalcommits.org/) for your commit

messages.\*\* This format is used to generate changelogs and ensures

consistency and better filtering. Since this is a mono repository the

convention ensures commit messages can be easily mapped to their

sub-project. Also, see existing commits for reference

([example](https://github.com/thi-ng/umbrella/commit/ebbc4910f64c90df7bb93010a75307df51c80b6e)).

The Conventional Commits classifiers/prefixes used in this project are:

- `feat` - new feature

- `fix` - bug fixes

- `refactor` - refactoring

- `test` - testing related

- `perf` - any type of optimization (not just performance)

- `build` - build/dependency related updates

- `doc` - documentation related only (e.g. readme, doc strings...)

- `chore` - unclassified chores

- `minor` - usually fixed typos (unless it qualifies as bug fix)

### Run & add tests

I'm heavily using the Node REPL during development and do much of my

testing as part of that workflow. Still, I'm aware that this is no full

replacement for a large suite of tests, therefore most packages do have

a varying (but growing) number of unit tests. If you're adding a new

feature (or think you've fixed a bug), please add some related tests (if

possible) too for extra brownie points. Either add a new file under a

project's `/test` dir or add/edit one of the existing test cases in

there.

Tests can be run via:

```bash

lerna run test --scope @thi.ng/<package-name>

# or

(cd packages/<package-name> && yarn test)

# or all tests (from repo root)

yarn test # also builds all packages first

yarn test:only # assumes all packages have been build already

```

### Push to your feature branch

```

git push origin feature/my-feature

```

### Create new Pull Request

Go to your fork on GH and create a PR. If there's no prior issue related

to the PR, please make sure you explain its purpose.

## Code style guide

### Project layout

Unless the package is very small, all larger ones in this repo share this

pattern:

- `/src/index.ts` - only used for re-exports

- `/src/api.ts` - interfaces, enums, type aliases & module global consts

definitions

- `/test` - Mocha unit tests

In larger packages (e.g.

[@thi.ng/transducers](https://github.com/thi-ng/umbrella/tree/master/packages/transducers))

topically related files are grouped in sub-folders under `/src`.

### Imports

To encourage small(er) file sizes of production artifacts, most source

files are organized to only contain a small number of related

functions/classes. Package internal imports MUST always refer to the

actual source file, whereas imports from other packages MUST only use

the package name. This is because of the way each package is built and

output in 3 different module formats (ES6, CJS, UMD).

### Sorted imports

Please ensure you're updating the list of imports in changed files to be

sorted by package name. In VSCode it's as easy as hitting `Shift + Alt +

O` or choosing "Organize imports" from the command palette.

### No default exports

Nuff said. They're potentially problematic in terms of refactoring and

too cause inconsistencies compared to the above named import pattern.

### No 3rd party dependencies

\*\*This does NOT apply to examples, only code in source packages\*\*

Unless absolutely warranted. Yes, this is somewhat a case of

\_Not-Invented-Here\_, but here done for reasons of sanity & clarity, not

to prove a point. If you plan to submit code with 3rd party deps, please

get in touch first and explain why it's necessary (in your humble opinion).

### Intra-repo dependencies

Please use your best judgment before introducing a new dependency on

another package within this repo and remember that even though these

packages are developed under one "umbrella", the aim is NOT to form a

tightly coupled framework. In general, it's absolutely fine to depend on

any of the "low level" packages, e.g.

- [@thi.ng/api](https://github.com/thi-ng/umbrella/tree/develop/packages/api)

- [@thi.ng/checks](https://github.com/thi-ng/umbrella/tree/develop/packages/checks)

- [@thi.ng/equiv](https://github.com/thi-ng/umbrella/tree/develop/packages/equiv)

- [@thi.ng/compare](https://github.com/thi-ng/umbrella/tree/develop/packages/compare)

- [@thi.ng/compose](https://github.com/thi-ng/umbrella/tree/develop/packages/compose)

- [@thi.ng/arrays](https://github.com/thi-ng/umbrella/tree/develop/packages/arrays)

- [@thi.ng/strings](https://github.com/thi-ng/umbrella/tree/develop/packages/strings)

- [@thi.ng/defmulti](http://github.com/thi-ng/umbrella/tree/develop/packages/defmulti) etc.

...since these are purely existing for providing general plumbing and

are primarily meant for wide re-use. However, consider if adding a

dependency on one of the larger packages (e.g.

[@thi.ng/transducers](https://github.com/thi-ng/umbrella/tree/develop/packages/transducers),

[@thi.ng/geom](https://github.com/thi-ng/umbrella/tree/develop/packages/geom))

is absolutely required / beneficial.

If in doubt, please ask first...

### No exported `const enums`

It has been [brought to my

attention](https://github.com/thi-ng/umbrella/issues/154) (thanks to

@Bnaya) that exported `const enums` negatively interfere with some

downstream workflows related to Babel transpilation and TypeScript's

`isolatedModules` compilation feature. For that reason, that latter

option is now enabled for all packages and exported `const enum`s are

NOT to be used anymore in this project. The only exceptions are packages

where `const enums` are used internally. For all others, we have

reverted to using normal `enum`s, but might introduce alternatives in

the future...

### Arrow functions preferred

Again, this is highly subjective - but unless a function requires

overrides, please use arrow functions for succinctness and avoidance of

potential scoping issues.

```ts

const add = (a: number, b: number) => a + b;

// vs.

function add(xs: Iterable<number>): number;

function add(a: number, b: number): number;

function add(a: any, b?: number): number {

if (typeof a === "number") {

return a + b;

}

let sum = 0;

for (let x of a) {

sum += x;

}

return sum;

};

```

### Function / ctor arguments

If a function or constructor takes multiple optional arguments, please

consider using a typed options argument instead of positional args. This

pattern is used in various existing packages already and involves

introducing a new `interface` for the options (see [naming

conventions](#naming-conventions)), e.g.:

```ts

import type { Fn, Comparator } from "@thi.ng/api";

// type arg is optional / context specific

interface SortOpts<T> {

/\*\*

\* Key extractor

\*/

key: Fn<A, number>;

/\*\*

\* Optional comparator

\*/

cmp?: Comparator<number>;

}

const sort = <T>(coll: T[], opts: FooOpts<T>) => {

const cmp = opts.cmp || ((a, b) => a - b);

return coll.sort((a, b) => cmp(opts.key(a), opts.key(b)));

};

```

### Naming conventions

These are not fully set in stone, but there's been a recent effort

underway to unify naming conventions & patterns for several

aspects/groups of functions / types:

#### General naming

Prefer short (though not cryptic) names over

`highlyDescriptiveLongCompoundName` style variable (or function) names,

which completely destroy formatting and resulting code comprehension (a

kind of counter-effect of the supposedly more descriptive longer name).

- Packages: `lower-kebab-case`

- Classes: capitalized `CamelCase`

- Functions, class methods & Variables: lower-initial `camelCase`

- Exported constants: `UPPER\_SNAKE\_CASE`

- Enums:

- type name itself: capitalized `CamelCase`

- constants: `UPPER\_SNAKE\_CASE`

#### Interfaces

If the interface is primarily defining a set of operations, we prefix

its name with `I` to distinguish it from a data descriptor.

```ts

interface IBind<T> {

bind(opts: T): boolean;

unbind(): boolean;

}

interface BindOpts {

texID: string;

texOpts: TextureOpts;

}

interface TextureOpts {

filter: Filter;

wrap: WrapMode;

...

}

```

#### Factory functions

Not (yet?) used consistently, but in order to encourage a more function

driven coding style (regardless of using some OOP concepts internally),

functions which create some form of resource/object/class instance

\*should\* be using the `def` prefix (inspired by Clojure and other

Lisps). For classes, this means adding a factory function delegating to

the class constructor and potentially performing additional preparation

tasks. This is not just done for stylistic reasons, but also to work

around the limitation of not being able to provide overrides for class

ctors (however function overrides are supported, of course).

```ts

/\*\*

\* Returns a new {@link Particle} instance with given initial position

\*

\* @param pos - initial position

\*/

export const defParticle = (pos: Vec) => new Particle(pos);

export class Particle {

constructor(public pos: Vec) {}

}

// usage

const particles = [[0, 0], [1, 0], [2, 0]].map(defParticle);

// vs

const particles = [[0, 0], [1, 0], [2, 0]].map((p) => new Particle(p));

```

There're other situations (e.g.

[@thi.ng/geom](https://github.com/thi-ng/umbrella/tree/develop/packages/geom)

or

[@thi.ng/rstream](https://github.com/thi-ng/umbrella/tree/develop/packages/rstream))

where this naming convention is not making much sense, but the above is

the currently \*preferred\* approach. Not dogma!

Standalone factory functions are also favored over `static` class

methods (though some are currently still in use, soon to be refactored).

#### Options objects

As stated [above](#function--ctor-arguments), interfaces describing a

collection of optional function / ctor arguments should always be

suffixed with `Opts`.

### Doc strings

In 2019 we refactored doc strings across all 120+ packages to become an

early adopter of the [TSDoc](https://github.com/microsoft/tsdoc)

documentation standard and [API extractor](https://api-extractor.com/)

toolchain, both developed by Microsoft.

Whilst that tooling is still WIP, please familiarize yourself with the

available documentation tags and use the format when adding/updating doc

strings.

There's been some initial work done on generating a better documentation

site than the current [docs.thi.ng](https://docs.thi.ng), but currently

on hold until the TSDoc standard is finalized / stable...

More discussion & context can be found in [this

issue](https://github.com/thi-ng/umbrella/issues/174).

### Formatting

All source code is to be formatted with [Prettier](https://prettier.io)

and a config file is included in the repo root.

If you're using VSCode, I recommend installing the [Prettier

extension](https://github.com/prettier/prettier-vscode) and configuring

it to auto-format on save.

For others, the important rules are:

- 4 spaces, no tabs (sorry!)

- semicolons enabled

- unix line breaks

## Donations

This project has been in development since early 2016 (some packages

even older). If you would like to support the continued development &

ever-increasing maintenance effort of this project, please consider a

financial contribution (anything helps!) via any of the following

channels:

- [GitHub Sponsors](https://github.com/sponsors/postspectacular)

- [Patreon](https://www.patreon.com/thing\_umbrella)

- BTC: 132aMfzNypBPgEy4Lz2tPQsKGimixdFrsb

- LTC: LMyfhJoXTq62W9zvUBvk9o6pCDZJx12dPV

- ETH: 0x8530bD57cCfCD5e95939E5bA3d81D8c9C9581941