Tutorial for Writing a TypeScript Library

{{> ./components/analytics.html}} {{> ./components/header/header.html}}

Building a typescript library?

How to write a typescript library

Writing modular code is a good idea. And nothing is more modular than writing a library. How can you write a typescript library? Well, that's exactly what this website is about! This tutorial is from **mid 2017** and written for **Typescript 2.x**!

Step 1: Setup tsconfig.json

Create a project folder, in this tutorial we'll call it `typescript-library`. Then proceed to create a `tsconfig.json` in `typescript-library`. Your `tsconfig.json` file should look somewhat like this:

typescript-library/tsconfig.json

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "declaration": true,
    "outDir": "./dist"
  },
  "include": [
    "src/**/*"
  ]
}
```

Pretty much like a setup for a non-library project, but with one important addition: You need to add the "declaration: true" flag. This will generate the typings required for the consumers to use this library.

Step 2: Implement your library

Proceed as you would if you wouldn't write a library. Create a `src` folder and in that `src` folder all the files that make up your heart of your library.

For this demo, we'll setup a silly `hello-world.ts` file, that looks like so:

typescript-library/src/hello-world.ts

```
export namespace HelloWorld {
  export function sayHello() {
    console.log('hi')
  }
  export function sayGoodbye() {
    console.log('goodbye')
  }
}
```

Step 3: Create an index.ts file

Add an `index.ts` file to your `src` folder. The purpose of it is to export all the parts of the library you want to make available for consumers. In our case it would simply be:

typescript-library/src/index.ts

```
export {HelloWorld} from './hello-world'
```

The consumer would be able to use the library later on like so:

someotherproject/src/somefile.ts

```
import {HelloWorld} from 'hwrld'
HelloWorld.sayHello();
```

You see that we have a new name here, "hwrld", we haven't seen anywhere yet. What is this name? It's the name of the library you're gonna publish to npm also known as the package name!

Step 4: Configure package.json

The package name is what the consumer going to use in the imports later on, so I prefer to not have it too long and if possible with no hyphens since it's easiest just to type a string when declaring the library as an import (an the consumers editor doesn't have auto-import). If you have to type `import * as`

`typescript-library from 'my-crazy-library-name'` each time, it's not so pleasant. For this demo I have have chosen `hwrld` since it was still available on npm.

The package name is usually right at the top of the `package.json`, so the package.json. The whole `package.json` would look like so:

typescript-library/package.json

```
{
  "name": "hwrld",
  "version": "1.0.0",
  "description": "Can log \"hello world\" and \"goodbye world\" to the console!",
  "main": "dist/index.js",
  "types": "dist/index.d.ts"
}
```

If you don't have a package yet you can create one with `npm init` and it will guide you through the process. Remember, the package name you choose will be the one people later use in their imports, so choose wisely!

There's also one all-important flag in this package.json: You have to declare where to find the type declarations! This is done using "types": "dist/index.d.ts" Otherwise the consumer won't find your module!

Step 5: Configure .npmignore

When you look at your package when loading it from npm, you'll notice that it contains the dist AND the src folder. However, you don't really need the source folder, as the compiled files along with the typings (.d.ts files) live in the dist folder. Thus you can create an .npmignore file. In our case it's simple:

typescript-library/.npmignore

```
tsconfig.json
src
```

Step 6: Publish to npm

To publish your first version to npm run:

```
tsc
npm publish
```

Now you're all set to go! Consume your library anywhere you want by running:

```
npm install --save hwrld
```

and consume it using

```
import {HelloWorld} from 'hwrld'
HelloWorld.sayHello();
```

For subsequent releases, use the semvar principle. When you make a patch / bugfix to your library, you can run `npm version patch`, for new features run `npm version minor` and on breaking changes of your api run `npm version major`.

Check out the full source of the demo library on github: https://github.com/bersling/typescript-library-starter.

The above tutorial contains all the steps necessary to build & publish a working library. However, you should probably also include some unit tests and you might want to test the behavior of your library locally first, without publishing. Here are some more resources for this:

- How to unit-test your library
- How to set up a local consumer without publishing to npm
- How to make your library available as a system command

{{> ./components/article/article-footer.html}}