TypeScript Package Project Structure

Reusable npm library with TypeScript, ESM/CJS dual publishing, and type declarations.

#typescript #package #library #npm #esm

PNG PDF Copy Prompt

Project Directory

```
mypackage/
    package.json  Dual ESM/CJS ex…
    tsconfig.json  TypeScript conf…
    tsconfig.build.json  Build-specific …
    LICENSE
    README.md
    .gitignore
    .npmignore  Exclude src fro…
    CHANGELOG.md  Version history
  src/  Source code
      index.ts  Public API expo…
      core.ts  Main functional…
      types.ts  Type definitions
      errors.ts  Custom error cl…
    internal/  Private impleme…
        index.ts
        utils.ts
        parser.ts
  dist/  Build output
      index.js  ESM bundle
      index.cjs  CommonJS bundle
      index.d.ts  Type declaratio…
  tests/
      core.test.ts
      types.test.ts
```

Why This Structure?

Modern npm packages need to support both ESM and CommonJS consumers. This structure uses TypeScript with dual exports in `package.json`. The `src/` directory contains your code, `dist/` contains the build output with `.js`, `.cjs`, and `.d.ts` files.

Key Directories

`src/index.ts` - Public API—export only what users need

`src/types.ts` - Shared type definitions

`src/internal/` - Implementation details, not exported

`dist/` - Build output with ESM, CJS, and .d.ts

Dual ESM/CJS Exports

```
// package.json exports
{
  "name": "mypackage",
  "type": "module",
  "exports": {
    ".": {
      "import": "./dist/index.js",
      "require": "./dist/index.cjs",
      "types": "./dist/index.d.ts"
    }
  },
  "main": "./dist/index.cjs",
  "module": "./dist/index.js",
  "types": "./dist/index.d.ts",
  "files": ["dist"]
}
```

Getting Started

1. `npm init -y`
2. `npm add -D typescript tsup vitest`
3. Configure `exports` in package.json
4. `npm run build` with tsup
5. `npm publish`

When To Use This

- Building a reusable npm package
- Library consumed by both ESM and CJS projects
- Publishing to npm registry
- Need type declarations for consumers
- Internal company packages

Trade-offs

Dual publishing complexity - ESM + CJS requires careful configuration

Build tooling - Need tsup or similar for bundling

Testing exports - Must verify both ESM and CJS work

Best Practices

- Export only public API from `index.ts`
- Use `exports` field for Node.js 12.7+
- Include `types` field for TypeScript consumers
- Test both ESM and CJS imports
- Use `files` field to minimize package size

Naming Conventions

Package name - lowercase with hyphens (my-package)

Source files - camelCase or kebab-case (utils.ts)

Types - PascalCase exports (MyOptions, Config)