

TypeScript configuration



Contents >

Configuration files

Strict mode

noImplicitAny and *suppressImplicitAnyIndexErrors*

...

TypeScript is a primary language for Angular application development. It is a superset of JavaScript with design-time support for type safety and tooling.


Browsers can't execute TypeScript directly. Typescript must be "transpiled" into JavaScript using the *tsc* compiler, which requires some configuration.

This page covers some aspects of TypeScript configuration and the TypeScript environment that are important to Angular developers, including details about the following files:

- [tsconfig.json](#)—TypeScript compiler configuration.
- [typings](#)—Typescript declaration files.

Configuration files

A given Angular workspace contains several TypeScript configuration files. At the root level, there are two main TypeScript configuration files: a `tsconfig.json` file and a `tsconfig.base.json` file.

The `tsconfig.json` file is a ["Solution Style"](#)  TypeScript configuration file. Code editors and TypeScript's language server use this file to improve development experience. Compilers do not use this file.

The `tsconfig.json` file contains a list of paths to the other TypeScript configuration files used in the workspace.

tsconfig.json

```
{
  "files": [],
  "references": [
    {
      "path": "./tsconfig.app.json"
    },
    {
      "path": "./tsconfig.spec.json"
    },
    {
```



```
"path": "../projects/my-lib/tsconfig.lib.json"
```

```
}  
]  
}
```

The `tsconfig.base.json` file specifies the base TypeScript and Angular compiler options that all projects in the workspace inherit.

The TypeScript and Angular have a wide range of options which can be used to configure type-checking features and generated output. For more information, see the [Configuration inheritance with extends](#) section of the TypeScript documentation.

For more information TypeScript configuration files, see the official [TypeScript wiki](#). For details about configuration inheritance, see the [Configuration inheritance with extends](#) section.

The initial `tsconfig.base.json` for an Angular workspace typically looks like the following example.

tsconfig.base.json

```
{  
  "compileOnSave": false,  
  "compilerOptions": {  
    "baseUrl": "./",  
    "outDir": "./dist/out-tsc",  
    "sourceMap": true,  
    "declaration": false,  
    "downlevelIteration": true,  
    "experimentalDecorators": true,  
    "moduleResolution": "node",  
    "importHelpers": true,  
    "target": "es2015",  
    "module": "es2020",  
    "lib": [  
      "es2018",  
      "dom"  
    ]  
  }  
}
```

Strict mode

When you create new workspaces and projects, you have the option to use Angular's strict mode, which can help you write better, more maintainable code. For more information, see [Strict mode](#).

`noImplicitAny` and `suppressImplicitAnyIndexErrors`

TypeScript developers disagree about whether the `noImplicitAny` flag should be `true` or `false`. There is no correct answer and you can change the flag later. But your choice now can make a difference in larger projects, so it merits discussion.

When the `noImplicitAny` flag is `false` (the default), and if the compiler cannot infer the variable type based on how it's used, the compiler silently defaults the type to `any`. That's what is meant by *implicit any*.

When the `noImplicitAny` flag is `true` and the TypeScript compiler cannot infer the type, it still generates the JavaScript files, but it also **reports an error**. Many seasoned developers prefer this stricter setting because type checking catches more unintentional errors at compile time.

You can set a variable's type to `any` even when the `noImplicitAny` flag is `true`.

When the `noImplicitAny` flag is `true`, you may get *implicit index errors* as well. Most developers feel that *this particular error* is more annoying than helpful. You can suppress them with the following additional flag:

```
"suppressImplicitAnyIndexErrors": true
```



For more information about how the TypeScript configuration affects compilation, see [Angular Compiler Options](#) and [Template Type Checking](#).

TypeScript typings

Many JavaScript libraries, such as jQuery, the Jasmine testing library, and Angular, extend the JavaScript environment with features and syntax that the TypeScript compiler doesn't recognize natively. When the compiler doesn't recognize something, it throws an error.

Use [TypeScript type definition files](#) `—d.ts files—` to tell the compiler about the libraries you load.

TypeScript-aware editors leverage these same definition files to display type information about library features.

Many libraries include definition files in their npm packages where both the TypeScript compiler and editors can find them. Angular is one such library. The `node_modules/@angular/core/` folder of any Angular application contains several `d.ts` files that describe parts of Angular.

You don't need to do anything to get *typings* files for library packages that include `d.ts` files. Angular packages include them already.

lib.d.ts

TypeScript includes a special declaration file called `lib.d.ts`. This file contains the ambient declarations for various common JavaScript constructs present in JavaScript runtimes and the DOM.

Based on the `--target`, TypeScript adds *additional* ambient declarations like `Promise` if the target is `es6`.

By default, the target is es2015. If you are targeting es5, you still have newer type declarations due to the list of declaration files included:

tsconfig.json (lib excerpt)

```
"lib": [  
  "es2018",  
  "dom"  
]
```



Installable typings files

Many libraries—jQuery, Jasmine, and Lodash among them—do *not* include `d.ts` files in their npm packages. Fortunately, either their authors or community contributors have created separate `d.ts` files for these libraries and published them in well-known locations.

You can install these typings via npm using the `@types/*` [scoped package](#) and Typescript, starting at 2.0, automatically recognizes them.

For instance, to install typings for `jasmine` you run `npm install @types/jasmine --save-dev`.

target

By default, the target is es2015, which is supported only in modern browsers. You can configure the target to es5 to specifically support legacy browsers. [Differential loading](#) is also provided by the Angular CLI to support modern, and legacy browsers with separate bundles.

RESOURCES

[About](#)

[Resource Listing](#)

[Press Kit](#)

[Blog](#)

[Usage Analytics](#)

HELP

[Stack Overflow](#)

[Gitter](#)

[Report Issues](#)

[Code of Conduct](#)

COMMUNITY

[Events](#)

[Meetups](#)

[Twitter](#)

[GitHub](#)

[Contribute](#)

LANGUAGES

[简体中文版](#)

[正體中文版](#)

[日本語版](#)

[한국어](#)

Super-powered by Google ©2010-2020. Code licensed under an MIT-style License. Documentation licensed under CC BY 4.0.

Version 10.0.10-local+sha.84d1ba792b.