TypeScript Configuration

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—TypesScript declaration files.

{@a tsconfig}

tsconfig.json

Typically, you add a TypeScript configuration file called `tsconfig.json` to your project to guide the compiler as it generates JavaScript files.

For details about `tsconfig.json`, see the official [TypeScript wiki](http://www.typescriptlang.org/docs/handbook/tsconfig-json.html).

The Setup guide uses the following `tsconfig.json`:

This file contains options and flags that are essential for Angular applications.

{@a noImplicitAny}

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`.

The documentation setup sets the `noImplicitAny` flag to `true`. 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

The documentation setup sets this flag to `true` as well.

{@a typings}

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 need do nothing 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`.

Since the QuickStart is targeting `es5`, you can override the list of declaration files to be included:

"lib": ["es2015", "dom"]

Thanks to that, you have all the `es6` typings even when targeting `es5`.

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 could do `npm install @types/jasmine --save-dev`.

QuickStart identifies two *typings*, or `d.ts`, files:

- jasmine typings for the Jasmine test framework.
- node for code that references objects in the *nodejs* environment; you can view an example in the webpack page.

QuickStart doesn't require these typings but many of the samples do.