



TUTORIAL

How To Work With TypeScript in Visual Studio Code

VS Code

By [James Quick](#)

Last Validated on April 17, 2020 • Originally Published on December 12, 2019 • 19.5k

Introduction

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. Let's break down what exactly this means:

- **typed** - You can define variable, parameter, and return data types.
- **superset** - TypeScript adds some additional features on top of JavaScript. All valid JavaScript is valid TypeScript, but not the other way around.
- **compiles to plain JavaScript** - TypeScript cannot be run by the browser. So available tooling takes care of compiling your TypeScript to JavaScript for the browser to understand.

In this tutorial you will work with TypeScript in Visual Studio Code to explore the benefits of using them together.

Prerequisites

For this project, you will need:

- A working understanding of JavaScript. You can review the [How to Code in JavaScript](#) series for more information.
- Node.js installed locally, which you can do by following [How to Install Node.js and Create a Local Development Environment](#).
- [Visual Studio Code](#) (VS Code) downloaded and installed.

Step 1 — Installing and Compiling TypeScript

The first step toward working with TypeScript is to install the package globally on your computer. Install the `typescript` package globally by running the following command in your terminal:

```
$ npm install -g typescript
```

Next, run the following command to make a project directory:

```
$ mkdir typescript_test
```

Move into the newly created directory:

```
$ cd typescript_test
```

Now, you need to create a new TypeScript file. For reference, these end with the `.ts` extension type.

You can now open up VS Code and create a new file by clicking **File** and then **New File**. After that, save it by clicking **File** and then **Save As....** You can name this file `app.ts` to follow this tutorial. The filename is not important, but ensuring the filetype extension of `.ts` is.

The first line of this file should start with an `export {};` for VS Code to recognize it as a module.

Create a function that will print the first and last name from a `person` object:

```
app.ts

export {};

function welcomePerson(person) {
  console.log(`Hey ${person.firstName} ${person.lastName}`);
  return `Hey ${person.firstName} ${person.lastName}`;
}

const james = {
  firstName: "James",
  lastName: "Quick"
};
```

```
welcomePerson(james);
```

The problem with the code above is that there is no restriction on what can be passed to the `welcomePerson` function. In TypeScript, you can create interfaces that define what properties an object should have.

In the snippet below, there is an interface for a `Person` object with two properties, `firstName` and `lastName`. Then, the `welcomePerson` function was modified to accept only `Person` objects.

```
app.ts

export {};

function welcomePerson(person: Person) {
  console.log(`Hey ${person.firstName} ${person.lastName}`);
  return `Hey ${person.firstName} ${person.lastName}`;
}

const james = {
  firstName: "James",
  lastName: "Quick"
};

welcomePerson(james);

interface Person {
  firstName: string;
  lastName: string;
}
```

The benefit of this will become clear if you try to pass a string into the `welcomePerson` function.

For example, replacing `james`:

```
welcomePerson(james);
```

With `'James'`:

```
welcomePerson('James');
```

Because we are working with a TypeScript file, VS Code will immediately provide you feedback letting you know that the function expects a `Person` object and not a string.

Output

```
$ Argument of type '"James"' is not assignable to parameter of type 'Person'.
```

Now that you have a working TypeScript file, you can compile it to JavaScript. To do this you need to call the function and tell it which file to compile. You can utilize the built-in terminal in VS Code to do this.

```
$ tsc app.ts
```

If you didn't fix the error before, you'll see an error output:

Output

```
$ app.ts:13:15 - error TS2345: Argument of type '"James"' is not assignable to paramete
```

Fix the error by passing the `Person` object in correctly instead of a string. Then compile again, and you'll get a valid JavaScript file.

Running an `ls` command in the terminal will display the files in our current path:

```
ls
```

You will see your original `ts` file and also a new `js` file:

Output

```
$ app.js
```

```
$ app.ts
```

Now, let's open the `app.js` file in VS Code.

```
app.js
```

```
"use strict";
exports.__esModule = true;
function welcomePerson(person) {
  console.log("Hey " + person.firstName + " " + person.lastName);
  return "Hey " + person.firstName + " " + person.lastName;
}
var james = {
  firstName: "James",
  lastName: "Quick"
};
welcomePerson(james);
```

Notice that the template literal strings, which are an ES6 feature, were compiled to simple string concatenation from ES5. We'll come back to this shortly.

To verify that this worked, you can now run the JavaScript directly using Node in your terminal:

```
$ node app.js
```

You will see a name printed to the console:

```
Output
$ Hey James Quick
```

Step 2 — Creating a TypeScript Config File

So far, you've compiled one file directly. This is great, but in a real world project, you might want to customize how all files are compiled. For instance, you might want to have them be compiled to ES6 instead of ES5. To do this, you need to create a TypeScript configuration file.

To create a TypeScript configuration file, you can run the following command (similar to an `npm init`):

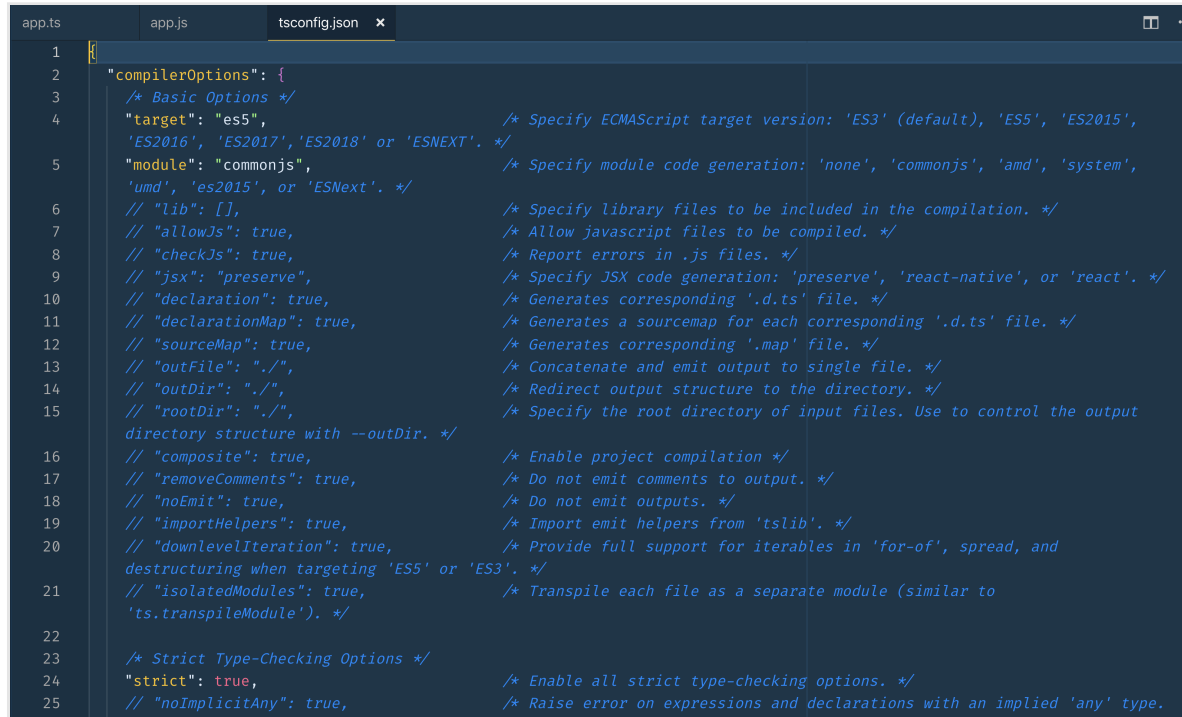
```
$ tsc --init
```

You will receive this output:

Output

```
$ message TS6071: Successfully created a tsconfig.json file.
```

Open your new `tsconfig.json` file and you'll see lots of different options, most of which are commented out.



```
1 {
2   "compilerOptions": {
3     /* Basic Options */
4     "target": "es5", /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015',
5                        'ES2016', 'ES2017', 'ES2018' or 'ESNEXT'. */
6     "module": "commonjs", /* Specify module code generation: 'none', 'commonjs', 'amd', 'system',
7                            'umd', 'es2015', or 'ESNext'. */
8     // "lib": [], /* Specify library files to be included in the compilation. */
9     // "allowJs": true, /* Allow javascript files to be compiled. */
10    // "checkJs": true, /* Report errors in .js files. */
11    // "jsx": "preserve", /* Specify JSX code generation: 'preserve', 'react-native', or 'react'. */
12    // "declaration": true, /* Generates corresponding '.d.ts' file. */
13    // "declarationMap": true, /* Generates a sourcemap for each corresponding '.d.ts' file. */
14    // "sourceMap": true, /* Generates corresponding '.map' file. */
15    // "outFile": "./", /* Concatenate and emit output to single file. */
16    // "outDir": "./", /* Redirect output structure to the directory. */
17    // "rootDir": "./", /* Specify the root directory of input files. Use to control the output
18                        directory structure with --outDir. */
19    // "composite": true, /* Enable project compilation */
20    // "removeComments": true, /* Do not emit comments to output. */
21    // "noEmit": true, /* Do not emit outputs. */
22    // "importHelpers": true, /* Import emit helpers from 'tslib'. */
23    // "downlevelIteration": true, /* Provide full support for iterables in 'for-of', spread, and
24                                destructuring when targeting 'ES5' or 'ES3'. */
25    // "isolatedModules": true, /* Transpile each file as a separate module (similar to
26                               'ts.transpileModule'). */
27
28    /* Strict Type-Checking Options */
29    "strict": true, /* Enable all strict type-checking options. */
30    // "noImplicitAny": true, /* Raise error on expressions and declarations with an implied 'any' type.
31
32  }
```

You might have noticed there is a setting called `"target"` which is set to `"es5"`. Change that setting to `"es6"`.

With these changes made to `tsconfig.json`, run `tsc` in your terminal:

```
$ tsc
```

Note: Unlike before, you are not specifying an input file. The [official documentation](#) points out: "When input files are specified on the command line, `tsconfig.json` files are ignored."

Now, open up the newly created JavaScript file:

```
app.js

"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
function welcomePerson(person) {
    console.log(`Hey ${person.firstName} ${person.lastName}`);
    return `Hey ${person.firstName} ${person.lastName}`;
}
const james = {
    firstName: "James",
    lastName: "Quick"
};
welcomePerson(james);
```

Notice that the template literal string was left alone, proving that your TypeScript was compiled successfully to ES6.

Another thing you can change is where your JavaScript files are stored after being created. This setting can be specified in `"outDir"`.

Try deleting "outDir", and then start typing it in again. VS Code is providing you IntelliSense for which properties you can set in a TypeScript config file.

```

1 {
2   "compilerOptions": {
3     /* Basic Options */
4     "target": "es6" /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016', 'ES2017', 'ES2018' or 'ESNEXT'. */,
5     "module": "commonjs" /* Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es2015', or 'ESNext'. */,
6     // "lib": [], /* Specify library files to be included in the compilation. */
7     // "allowJs": true, /* Allow javascript files to be compiled. */
8     // "checkJs": true, /* Report errors in .js files. */
9     // "jsx": "preserve", /* Specify JSX code generation: 'preserve', 'react-native', or 'react'. */
10    // "declaration": true, /* Generates corresponding '.d.ts' file. */
11    // "declarationMap": true, /* Generates a sourcemap for each corresponding '.d.ts' file. */
12    // "sourceMap": true, /* Generates corresponding '.map' file. */
13    // "outFile": "./", /* Concatenate and emit output to single file. */
14    "out": "Redirect output structure to the directory. */",
15    // "outDir" Redirect output structure to the directory. x |les. Use to control the output directory structure
16    with "outFile"
17    // "preserveWatchOutput" + Enable project compilation */
18    // "noImplicitUsesStrict" + Do not emit comments to output. */
19    // "noUnusedParameters" + Do not emit outputs. */
20    // "sourceRoot" + Import emit helpers from 'tslib'. */
21    // "noErrorTruncation" + Provide full support for iterables in 'for-of', spread, and destructuring when targeting
22    'ES5' traceResolutio
23    // "moduleResolution" + Transpile each file as a separate module (similar to 'ts.transpileModule'). */
24    // "noImplicitReturns"
25    // "strictFunctionTypes"
26    "strict": "strict"
27    "strictFunctionTypes"
28    "strictFunctionTypes"
29    "strictFunctionTypes"
30    "strictFunctionTypes"
31    "strictFunctionTypes"
32    "strictFunctionTypes"
33    "strictFunctionTypes"
34    "strictFunctionTypes"
35    "strictFunctionTypes"
36    "strictFunctionTypes"
37    "strictFunctionTypes"
38    "strictFunctionTypes"
39    "strictFunctionTypes"
40    "strictFunctionTypes"
41    "strictFunctionTypes"
42    "strictFunctionTypes"
43    "strictFunctionTypes"
44    "strictFunctionTypes"
45    "strictFunctionTypes"
46    "strictFunctionTypes"
47    "strictFunctionTypes"
48    "strictFunctionTypes"
49    "strictFunctionTypes"
50    "strictFunctionTypes"
51    "strictFunctionTypes"
52    "strictFunctionTypes"
53    "strictFunctionTypes"
54    "strictFunctionTypes"
55    "strictFunctionTypes"
56    "strictFunctionTypes"
57    "strictFunctionTypes"
58    "strictFunctionTypes"
59    "strictFunctionTypes"
60    "strictFunctionTypes"
61    "strictFunctionTypes"
62    "strictFunctionTypes"
63    "strictFunctionTypes"
64    "strictFunctionTypes"
65    "strictFunctionTypes"
66    "strictFunctionTypes"
67    "strictFunctionTypes"
68    "strictFunctionTypes"
69    "strictFunctionTypes"
70    "strictFunctionTypes"
71    "strictFunctionTypes"
72    "strictFunctionTypes"
73    "strictFunctionTypes"
74    "strictFunctionTypes"
75    "strictFunctionTypes"
76    "strictFunctionTypes"
77    "strictFunctionTypes"
78    "strictFunctionTypes"
79    "strictFunctionTypes"
80    "strictFunctionTypes"
81    "strictFunctionTypes"
82    "strictFunctionTypes"
83    "strictFunctionTypes"
84    "strictFunctionTypes"
85    "strictFunctionTypes"
86    "strictFunctionTypes"
87    "strictFunctionTypes"
88    "strictFunctionTypes"
89    "strictFunctionTypes"
90    "strictFunctionTypes"
91    "strictFunctionTypes"
92    "strictFunctionTypes"
93    "strictFunctionTypes"
94    "strictFunctionTypes"
95    "strictFunctionTypes"
96    "strictFunctionTypes"
97    "strictFunctionTypes"
98    "strictFunctionTypes"
99    "strictFunctionTypes"
100   }
101 }

```

You can change your "outDir" from the current directory to a dist directory like so:

tsconfig.json

```
"outDir": "./dist"
```

After compiling again (`tsc`), notice that your output JavaScript file is indeed located inside of a `dist` directory.

You can use the `cd` and `ls` commands in your terminal to explore the contents of the `dist` directory:

```
$ cd dist  
$ ls
```

You will see your compiled JavaScript file in the new location:

```
Output  
$ app.js
```

Step 3 — Exploring TypeScript in Modern Front-End Frameworks

TypeScript has gained more and more popularity over the last couple of years. Here's a couple of examples of how it is used in modern front-end frameworks.

Angular CLI

Angular CLI projects come preconfigured with TypeScript. All of the configuration, linting, etc. is built in by default. Create an Angular CLI project and take a look around. This is a great way to see what TypeScript looks like in a real app.

Create React App 2

Create React App doesn't come with TypeScript by default, but with the latest version, it can be configured that way. If you're interested in learning how to use TypeScript with Create React App, check out the [\[Using Create React App v2 and TypeScript\]](#)

(<https://www.digitalocean.com/community/tutorials/using-create-react-app-v2-and-typescript>) article.

Vue CLI 3

Vue CLI projects can be configured to use TypeScript when creating a new project. For more details, you can check out the [Vue Docs](#).

Conclusion

In this tutorial, you explored using TypeScript with VS Code. TypeScript allows you to generate higher quality JavaScript that can provide more confidence when shipping to production. As you can tell, VS Code is well equipped to help you write TypeScript, generate configurations, and so on.

Was this helpful?

Yes

No



[Report an issue](#)

About the authors



[James Quick](#)

Developer. Speaker. Teacher. Developer Advocate at Auth0.

Still looking for an answer?



Ask a question



Search for more help

RELATED

Code Formatting with Prettier in Visual Studio Code

 [Tutorial](#)

How To Use Visual Studio Code for Remote Development via the Remote-SSH Plugin

 [Tutorial](#)

Comments

1 Comment

B

I

Leave a comment...

Sign In to Comment

0

ThomasMos01

April 21, 2020

thanks.
you helped me)))

Reply

Report

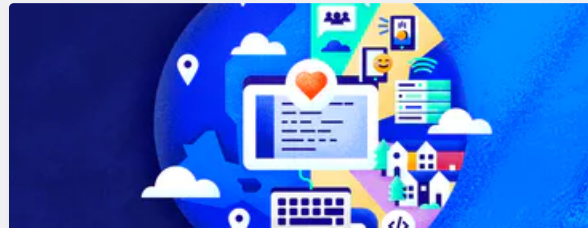


This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.



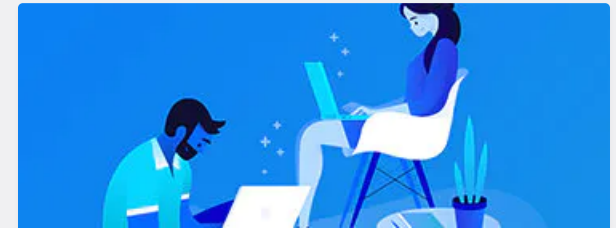
GET OUR BIWEEKLY NEWSLETTER

Sign up for Infrastructure as a Newsletter.



HUB FOR GOOD

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.



BECOME A CONTRIBUTOR

You get paid; we donate to tech nonprofits.

[Featured on Community](#) [Kubernetes Course](#) [Learn Python 3](#) [Machine Learning in Python](#) [Getting started with Go](#) [Intro to Kubernetes](#)

[DigitalOcean Products](#) [Virtual Machines](#) [Managed Databases](#) [Managed Kubernetes](#) [Block Storage](#) [Object Storage](#) [Marketplace](#) [VPC](#) [Load Balancers](#)

Welcome to the developer cloud

DigitalOcean makes it simple to launch in the cloud and scale up as you grow – whether you're running one virtual machine or ten thousand.



© 2021 DigitalOcean, LLC. All rights reserved.

Company

[About](#)
[Leadership](#)
[Blog](#)
[Careers](#)
[Partners](#)
[Referral Program](#)
[Press](#)
[Legal](#)
[Security & Trust Center](#)

Products

[Pricing](#)
[Products Overview](#)
[Droplets](#)
[Kubernetes](#)
[Managed Databases](#)
[Spaces](#)
[Marketplace](#)
[Load Balancers](#)
[Block Storage](#)
[API Documentation](#)
[Documentation](#)
[Release Notes](#)

Community

[Tutorials](#)
[Q&A](#)
[Tools and Integrations](#)
[Tags](#)
[Product Ideas](#)
[Write for DigitalOcean](#)
[Presentation Grants](#)
[Hatch Startup Program](#)
[Shop Swag](#)
[Research Program](#)
[Open Source](#)
[Code of Conduct](#)

Contact

[Get Support](#)
[Trouble Signing In?](#)
[Sales](#)
[Report Abuse](#)
[System Status](#)