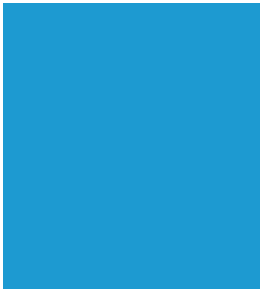


SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

Section



# How to use TypeScript with Node.js

June 14, 2021

Topics: [Node.js](#)



Typescript, a JavaScript superset, is gaining tremendous popularity among developers. It incorporates every JavaScript feature with supplementary traits such as static typing and type checking.

**SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.**[Read White Paper](#)

popular framework.

Node.js is great for making server-side applications, but it lacks some modern components such as type checking. A Node.js codebase can also be very hard to maintain.

Typescript supports these modern coding styles, such as static typing and type checking. It is made to build extensive and high-level applications. That's why it would be great to have Typescript as the primary language to support Node.js functionality.

This allows you to write server-side-based applications with strong type checking, which allows you to avoid runtime type errors and other Typescript advantages and take full advantage of Node.js.

To benefit from these vital Node.js features, you need to set up and configure your Typescript with Node.js runtime. This guide will teach you how to set up and run your Typescript application with Node.js and run some Node.js packages within your Typescript application.

## Prerequisites

To follow it tutorial along - the reader will need the following:

- [Node.js](#) installed on your computer.

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

Once Node.js is installed in the computer, run `node -v` to confirm if the installation was successful.

Just like you would have done with any application running on the Node.js runtime, create a project folder. Open a command line and change the directory to point to this new project folder.

To generate the Node.js package.json file, run `npm init`. This will introduce systematic questions about your project. This prompts the way you would set up the regular Node.js project. Alternatively, run `npm init -y` and override the package.json file with the default values.

```
$ npm init -y
Wrote to F:\Article testing\Typescript-with-Node.js\package.json:

{
  "name": "Typescript-with-Node.js",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

## Setting up Typescript

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.



Read White Paper



```
npm install -g typescript
```

The command above will install the Typescript compiler globally. This means any project you create on your computer can access Typescript dependencies without reinstalling the Typescript package when building another project.

Run `tsc --version` to confirm if the compiler is installed.

```
C:\Users\User>tsc --version  
Version 4.2.4
```

**Note:** Typescript code doesn't build directly on a browser (no browser will read Typescript directly). A browser will only read JavaScript code. To invoke any Typescript code, you need a compiler to transpile Typescript to browser executable JavaScript. Typescript will only help you in the development process so that you benefit from its features.

## Creating a Typescript config file

To initialize the Typescript project with Node.js, create a [tsconfig.json](#) file in the project root directory.

## SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

```
user@windows-kubeiso: ~/nodejs/article-testing/ts with some step by step  
$ tsc --init  
message TS6071: Successfully created a tsconfig.json file.
```

This file is the Typescript configuration file. It specifies typescript compile options.

```
tsconfig.json X  
tsconfig.json > {} compilerOptions  
1 {  
2   "compilerOptions": {  
3     /* Visit https://aka.ms/tsconfig.json to read more about this file */  
4  
5     /* Basic Options */  
6     // "incremental": true,           /* Enable incremental compilation */  
7     "target": "es5",                 /* Specify ECMAScript target version: 'ES3' (default), 'ES5', 'ES2015', 'ES2016',  
8     "module": "commonjs",            /* Specify module code generation: 'none', 'commonjs', 'amd', 'system', 'umd', 'es  
9     // "lib": [],                     /* Specify library files to be included in the compilation. */  
10    // "allowJs": true,                /* Allow javascript files to be compiled. */  
11    // "checkJs": true,               /* Report errors in .js files. */  
12    // "jsx": "preserve",              /* Specify JSX code generation: 'preserve', 'react-native', 'react', 'react-jsx' or  
13    // "declaration": true,            /* Generates corresponding '.d.ts' file. */  
14    // "declarationMap": true,         /* Generates a sourcemap for each corresponding '.d.ts' file. */
```

The `tsconfig.json` file has many options. It's good to know when to turn things on and off. TSC reads this file and uses these options to transpile Typescript into browser readable JavaScript.

Let's edit this config file to include the necessary option to run Typescript.

- `"target": "es6"` - one key thing you need to define is the version of JavaScript the compiler will output. If you need more advanced JavaScript features, such as using the arrow functions, `const`, and `let`,

**SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.**[Read White Paper](#)

JavaScript.

- "module": "commonjs" - this is the JavaScript module formatting system for structuring and organizing JavaScript code. This lets the compiler use module functions such as `require()` and objects such as `module.exports`.
- "rootDir": "./src" - a directory where the input Typescript files are stored. I named the folder as `src`.
- "outDir": "./dist" - this an output directory where the output structure of the compiled JavaScript will be saved. In this case, JavaScript code will be saved in a folder called `dist` in the root directory of the main project.
- "moduleResolution": "node" - a module import resolution algorithm that mimics the way the Node.js searches for modules in real-time.
- "strict": true - enable all JavaScript strict type-checking options.
- "esModuleInterop": true - `esModuleInterop` allows us to compile ES6 modules to `commonjs` modules.
- "exclude": [] - tells Typescript not to compile specified files or folders. In this case, you can tell the Typescript not to compile the `node_modules`

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

```
"exclude": [
  "./node_modules"
]
```

Check this [tutorial](#) to learn a few more options that you can include in your tsconfig.json file.

Here is an example of a tsconfig.json file option.

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "commonjs",
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "moduleResolution": "node",
    "esModuleInterop": true,
  },
  "exclude": [
    "./node_modules"
  ]
}
```

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.



Read White Paper



Go ahead and create the folder `src` in your project directory. `src` hosts the Typescript files as explained earlier. Inside `src`, create an `index.ts` file and start writing some Typescript code. While writing down Typescript, it is advisable to have a Typescript compiler installed in your project's local dependencies.

Run `npm install -D typescript` to do so.

```
function sum (num1:number, num2:number){  
    return num1 + num2;  
}  
console.log(sum(8,4))
```

Let's see how we can execute the above Typescript using Node.js. Run `tsc` to build Typescript into JavaScript. This will build, compile and output JavaScript code into the path you specified in "outDir": `./dist` of the `config.json` file.

```
JS index.js x  
dist > JS index.js > ...  
1  "use strict";  
2  function sum(num1, num2) {  
3    |   return num1 + num2;  
4  }  
5  console.log(sum(8, 4));  
6  |
```



SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

```
$ node dist/index.js  
12
```

Running the above commands every single time to compile and run in a development environment can be annoying. To make this process easier, you need to install a package called `ts-node`.

```
npm install -D ts-node
```

`Ts-node` allows us to point to a Typescript file. It will run `.ts`, compile it and run it with Node.js for us.

When using `Ts-node`, make sure you have Typescript installed in your local project. To install it, run `npm install -D typescript`.

Go ahead and configure your `package.json` script tag and start the build command as shown below.

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

You can now run `npm start` to execute the `index.ts`. And when you delete the `dist` folder with the compiled `index.js` then still run `npm start`, you will get the same results.

```
$ npm start  
  
> Typescript-with-Node.js@1.0.0 start F:\Article testing\Typescript-with-Node.js  
> ts-node ./src/index.ts  
  
12
```

## Using Typescript with packages

The climax of using Typescript and Node.js is the ability to utilize the open-source NPM packages and frameworks. Let me demonstrate how we can use Typescript to start interacting with NPM packages.

This example will set up Typescript With Express.js. You should install [Express.js](#) from the NPM registry.

To do so, run `npm install express`.

**Note:** Always make sure you have [Node.js types checking package](#) installed whenever writing Typescript using Node.js.

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

Read White Paper



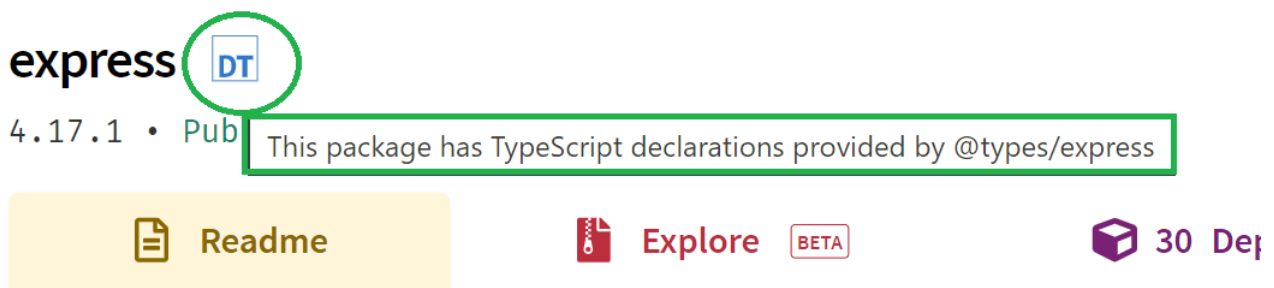
node.js packages are written in JavaScript and not typescript. To get the type definitions for its packages, you need to install third-party packages called @types.

For example, to use [Express type definitions](#), install @types/express by running:

```
npm install -D @types/express
```

This will give you access to type definitions for Express.

Not all packages have @types. Package with types has the following NPM @types tag.



Let's set a simple Express server using Typescript.

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

```
import express, {Request,Response,Application} from 'express';
```

- Initialize Express inside Typescript.

```
const app:Application = express();
```

- Set the server port.

```
const PORT = process.env.PORT || 8000;
```

- Set an endpoint/route.

```
app.get("/", (req:Request, res:Response):void => {  
  res.send("Hello Typescript with Node.js!")  
});
```

- Listen to the server port.

```
app.listen(PORT, ():void => {  
  console.log(`Server Running here 🖱️ https://localhost:${PORT}`);  
});
```

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.



Read White Paper



to you as defined in the `res.send()`.

When developing an extensive application, it's advisable to hook your project with watch parameters that will help you restart your server whenever you make and save changes to your code structure.

There are two main options:

1. Ts-node-dev - [Ts-node-dev](#) watches `.ts` files, and whenever you make a change, it will restart the server for you.

Run the command below to install it.

```
npm install -D ts-node-dev
```

To use it, modify the `package.json` script tag as shown below.

```
"scripts": {  
  "dev": "ts-node-dev --respawn ./src/index.ts"  
}
```

Then run `npm run dev`, and your server will be watched every time you make changes in your code.

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.



Read White Paper



Server Running here <https://localhost:8000>

2. Nodemon - [Nodemon](#) works the same as Ts-node-dev . It is a Node.js package used to watch .js files when creating a server. Whenever you make a change and save the server file, Nodemon will automatically restart the server for you.

Run the command below to install it.

```
npm install -D nodemon
```

To use it in your project, edit the package.json script tag as follows:

```
"scripts": {  
  "dev": "nodemon ./src/index.ts"  
}
```

Then run `npm run dev`.

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.

[Read White Paper](#)

```
[nodemon] watching extensions: ts,json  
[nodemon] starting `ts-node ./src/index.ts`  
Server Running here https://localhost:8000  
[nodemon] restarting due to changes...  
[nodemon] restarting due to changes...  
[nodemon] starting `ts-node ./src/index.ts`
```

## Conclusion

Typescript will help you in the development process so that you don't struggle with bugs and minor errors. It catches errors during compilation and before code runtime.

This gives you an eye ahead to figure out and fix runtime errors. This will improve your development speed, especially when building an extensive application.

Typescript has become a [popular language](#) among JavaScript developers. Check this [tutorial](#) and learn how to get started with Typescript.

Happy coding!

---

Peer Review Contributions by: [Mohan Raj](#)

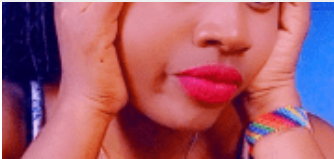
---

## About the author

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.



[Read White Paper](#)



enjoyable. A combination of Frontend and Backend development makes her a joyful upcoming software developer.

This article was contributed by a student member of Section's Engineering Education Program. Please report any errors or inaccuracies to [enged@section.io](mailto:enged@section.io).

## Want to learn more about the EngEd Program?

Discover Section's community-generated pool of resources from the next generation of engineers.

[Learn more →](#)



SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.



Read White Paper



## Join our Slack community



Add to Slack

---

### Company

About

Careers

Legals

### Resources

Blog

Case Studies

Content Library

Solution Briefs

Partners

Changelog

### Support

Docs

Community Slack

SOLVING THE EDGE PUZZLE: Your journey to the Edge begins here.




Read White Paper



Section supports many open source projects including:

 [varnish cache logo](#)

 [cloud native computing foundation logo](#)

 [the linux foundation logo](#)

 [IETF edge logo](#)



© 2021 Section

[Privacy Policy](#) [Terms of Service](#)

More than one instance of Sumo is attempting to start on this page. Please check that you are only loading Sumo once per page.