**TYPESCRIPT**

**What is Typescript :**

TypeScript is an Open Source [Object Oriented programming language](https://www.geeksforgeeks.org/introduction-of-object-oriented-programming/) developed and maintained by Microsoft Corporation. TypeScript is a strongly typed language and its first version was introduced in 2012. It is a Strict Super Set of JavaScript, which means anything that is implemented in JavaScript can be implemented using TypeScript along with the choice of adding enhanced features (every existing JavaScript Code is a valid TypeScript Code). As TypeScript code is converted to JavaScript code it makes it easier to integrate into JavaScript projects. It is designed mainly for large Scale projects.

TypeScript ===>>> JavaScript + Type + Some Added Features

**Features Of Typescript:**

1. Static Type Checking (Optional) – Like other high level programming languages like Java, C etc. TypeScript does provide static type checking unlike JavaScript. Although static typing requires some extra steps while writing code but it has its own advantages. With TypeScript, we can check and assign variables, parameters and function types. It is completely Optional and helps us find and prevent bugs. Also helps make code more readable and descriptive.
2. Class Based Objects – Another huge advantage is the use of Classes which provides the ability to use true object oriented programming in our applications and prevents use of prototype based objects. It also provides encapsulation, inheritance and modifiers.
3. Modularity – It helps make the code more modular.
4. ES6 Features – Support for ES6 features is also one of the main reasons for its popularity.
5. Syntax – TypeScript provides syntax which is closer to java and other high level languages (Syntactical Sugaring).

**Setting up Typescript Development Environment:**

*1. Create the project folder structure*

The project folder is the root of your development application. You can name the folder anything you like. I’m going to use *typescript-dev-env.*

mkdir typescript-dev-env

Create the new folder on disk and make it the current folder by doing a change directory (CD) into the new folder.

cd typescript-dev-env

Next, create a subfolder named src in the project folder.

mkdir src

Then create another subfolder named dist in the project folder.

mkdir dist

The folder structure should look like this:

*typescript-dev-env*

*dist/*

*src/*

*2. Create the project configuration file*

Create the *project.json* file.

npm init -y

The *project.json* file manages all the npm packages used in the project. It stores all the package names and versions that are needed for both development and runtime use. Plus, it allows creating scripts to execute complex commands. We’ll add one of those scripts next.

*3. Add a “start” script to start up the development environment*

In the *package.json* file, add a script to the *scripts* key. Insert the bolded code right before the line with "test". Don’t forget the comma at the end.

"scripts": {

"start": "webpack-dev-server --mode development",

"build": "webpack --mode production",

"test": "echo \"Error: no text specified\" && exit 1"

},

The start script will allow us to easily start-up the development environment using webpack-dev-server (to be installed later). When started, it will watch for changes in your source files and, when they are saved, will build and refresh the browser so you can see your changes in action quickly.

*4. Install TypeScript and Webpack*

Install all the needed packages using npm.

npm install typescript ts-loader webpack webpack-dev-server webpack-cli --save-dev

This command installs the TypeScript *transpiler* (tsc.exe) so that the Typescript files you create can be turned into JavaScript. This is needed because, at this time, the browsers only understand JavaScript. So TypeScript source files must be converted into JavaScript files through a process called *transpiling*. It is similar to compiling except that instead of creating object code from source, it creates another source file in JavaScript which the browsers do understand.

The *ts-loader* allows loading TypeScript files.

*Webpack* is a JavaScript “bundler” that takes multiple JavaScript files and “bundles” them together into a single JavaScript file.

The *webpack-dev-server* provides a continuous runtime environment so that changes to your TypeScript files will automatically be transpiled, bundled, and the browser refreshed.

This *webpack-cli* is the command line interface to the webpack component which is used in development.

*5. Create the webpack configuration file*

Webpack needs a configuration file to operate. Create a file named *webpack.config.js* in the project root and copy the following code into it.

const path = require('path');

module.exports = {

entry: './src/main.ts',

devtool: 'inline-source-map',

devServer: {

contentBase: './dist'

},

module: {

rules: [

{

test: /\.tsx?$/,

use: 'ts-loader',

exclude: /node\_modules/

}

]

},

resolve: {

extensions: [ '.tsx', '.ts', '.js' ]

},

output: {

filename: 'bundle.js',

path: path.resolve(\_\_dirname, 'dist')

}

};

This configuration indicates that there should be an *main.ts* file in the *src* folder under the project root. It also bundles all the JavaScript files into a single file called *bundle.js* and creates it in the *dist* folder under the project root.

*We will be running in development mode. In development mode, webpack does NOT create a physical bundle.js file in the dist folder so don’t be worried that it is not there. To create the bundle.js file in the dist folder, run npm run build.*

*6. Create the Typescript configuration file*

The Typescript configuration file is called *tsconfig.json* in the project root.

tsc --init.

This command uses the TypeScript transpiler program to create *tsconfig.json* configuration file for TypeScript in the project root folder.

The tsconfig.json file controls what level of JavaScript is generated from the TypeScript source and various other configuration options.

{

"compilerOptions": {

"outDir": "./dist/",

"sourceMap": true,

"noImplicitAny": true,

"module": "es6",

"target": "es5",

"jsx": "react",

"allowJs": true

}

}

This configuration causes the generated files to be stored in the dist folder.

This provides all we need to do in setting up the development environment. The last few steps will create test files in the environment and give it a try.

*7. Create index.html*

We start by creating the *index.html* file. It should be in the *dist* folder and should look like this:

<!doctype html>

<html lang="en">

<head>

<title></title>

</head>

<body>

<!-- put your html here -->

<script src="bundle.js"></script>

</body>

</html>

The <script src="bundle.js"> tag is placed at the end of the <body> tag to make sure all the html is loaded before any JavaScript accesses it. *Bundle.js* is the name of the bundle created from the TypeScript files.

*8. Create the file hello-world-in-typescript.class.ts*

In the src folder, create this TypeScript class that shows some of the features of TypeScript including defining a class. This class will construct a message and add it to the body of the document object model (DOM) dynamically.

The class has a single method called *HelloWorld()* which simply creates an <h1> tag, puts a string into it, and adds it to the body of the page. The method is marked *static* so it can be called without creating an instance of the HelloWorldInTypescript class. The call will be made in the *main.ts* file.

export class HelloWorldInTypescript {

static HelloWorld() {

let h1 = document.createElement("h1");

h1.innerText = "Hello, world in Typescript";

document.body.appendChild(h1);

}

}

*9. Create the file main.ts*

Create a new file called *main.ts* in the *src* folder.

The *webpack.config.js* file makes *main.ts* the starting point for our application. It similar to the *main()* method in C#.

It starts by importing the *HelloWorldInTypescript* class from the *hello-world-in-typescript.class.ts* file. This permits use of the HelloWorldInTypescript class within the main.ts file. Then it call the static *HelloWorld()* function of th class.

import { HelloWorldInTypescript }

from './hello-world-in-typescript.class';

HelloWorldInTypescript.HelloWorld();

*10. Build and run*

Start the development environment.

npm run start

## **Features of TypeScript**

**TypeScript is just JavaScript.** TypeScript starts with JavaScript and ends with JavaScript. Typescript adopts the basic building blocks of your program from JavaScript. Hence, you only need to know JavaScript to use TypeScript. All TypeScript code is converted into its JavaScript equivalent for the purpose of execution.

**TypeScript supports other JS libraries.** Compiled TypeScript can be consumed from any JavaScript code. TypeScript-generated JavaScript can reuse all of the existing JavaScript frameworks, tools, and libraries.

J**avaScript is TypeScript**. This means that any valid .js file can be renamed to .ts and compiled with other TypeScript files.

**TypeScript is portable.** TypeScript is portable across browsers, devices, and operating systems. It can run on any environment that JavaScript runs on. Unlike its counterparts, TypeScript doesn’t need a dedicated VM or a specific runtime environment to execute.

### **TypeScript and ECMAScript**

The ECMAScript specification is a standardized specification of a scripting language. There are six editions of ECMA-262 published. Version 6 of the standard is codenamed "Harmony". TypeScript is aligned with the ECMAScript6 specification.