

JavaScript Development Environments

JavaScript

JavaScript and React (CRA)

RUN ALL TESTS WITH FILEWATCH

If you create your react app using `npx create-react-app my-react-app` then this all tests with filewatch is the default for the `npm test` script. From the terminal just enter the following command

```
npm test
```

DEBUG ALL TESTS

Add the following code to your `launch.config`

```
{
  "name": "Debug CRA Tests",
  "type": "node",
  "request": "launch",
  "runtimeExecutable": "${workspaceRoot}/node_modules/.bin/react-scripts",
  "args": ["test", "--runInBand", "--no-cache", "--watchAll=false"],
  "cwd": "${workspaceRoot}",
  "protocol": "inspector",
  "console": "integratedTerminal",
  "internalConsoleOptions": "neverOpen",
  "env": { "CI": "true" },
  "disableOptimisticBPs": true
}
```

DEBUG SINGLE TEST FILE

Typescript

Setup

JAVASCRIPT/TYPESCRIPT PROJECT STRUCTURE

Command	Details
<code>package.json</code>	Describes a project's top-level dependencies. These are packages that have been added to a project using <code>npm install</code>
<code>package-lock.json</code>	All package dependencies for the project
<code>tsconfig.json</code>	TypeScript compiler configuration

NODE PACKAGES

Package	Description
<code>typescript</code>	The typescript compiler
<code>jest</code>	JavaScript testing framework
<code>tsc-watch</code>	Run a script specified in <code>package.json</code>
<code>ts-jest</code>	Enable us to use Jest with TypeScript

TYPESCRIPT COMPILER OPTIONS (TSCONFIG.JSON)

Package	Description
<code>target</code>	The version of JavaScript to transpile to
<code>module</code>	The JavaScript module format. Some environments such as node do not support ES2015 modules so specifying <code>commonjs</code> tells the compiler to generate older module code
<code>lib</code>	Tell TypeScript that certain APIs will be available in the environment in which the code will run. Examples such as DOM's <code>document.querySelector</code>
<code>outDir</code>	The directory into which we will put the transpiled JavaScript.
<code>include</code>	The folders to look in for TypeScript files

The following listing shows an example `tsconfig.json` file.

Listing 1 `tsconfig.json`

```
{
  "compilerOptions": {
    "target": "ES2018",
    "outDir": "./dist",
    "noEmitOnError": true,
    "sourceMap": true,
    "module": "commonjs"
  }
}
```

Development Template

This section shows how to setup a TypeScript development environment with the following benefits.

- Automatically transpile TypeScript files as they change.
- Unit testing with Jest.
- Debugging individual unit tests with Jest Visual Studio Code Plugin.

PACKAGE.JSON

```
{
  "name": "tools",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "npx jest --watchAll",
    "start": "tsc-watch --onSuccess \" node dist/index.js\""
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "tsc-watch": "^4.2.3",
    "typescript": "^3.8.3"
  }
}
```

The bold lines specify scripts that can be run by npm. We have added a script called start that monitors files for change and executes the index.js when changed files have been compiled

Typescript – Review to delete

Debugging

If we want to debug in VSCode we need to add a folder called `.vscode` into which we add a file called `launch.json`

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
```

```
// For more information, visit: https://go.microsoft.com/fwlink/?link
id=830387
"version": "0.2.0",
"configurations": [
  {
    "type": "node",
    "request": "launch",
    "name": "Launch Program",
    "program": "${workspaceFolder}\\dist\\index.js",
  }
]
```

We can then run our debugger using F5 in visual studio code

Unit Testing

Unit testing with Jest consists of two parts. The first part is to setup a configuration file called `jest.config.js` at the root level of our project. The following is a good example.

```
module.exports = {
  "roots": ["src"],
  "transform": {"^.+\\.tsx?$": "ts-jest"}
}
```

Then we simply add tests in our source code folder. If we have a module called `adder.ts` as follows

```
export function add(a: number, b: number): number {
  return a+b;
}
```

We can create a test called `adder.test.ts` as follows

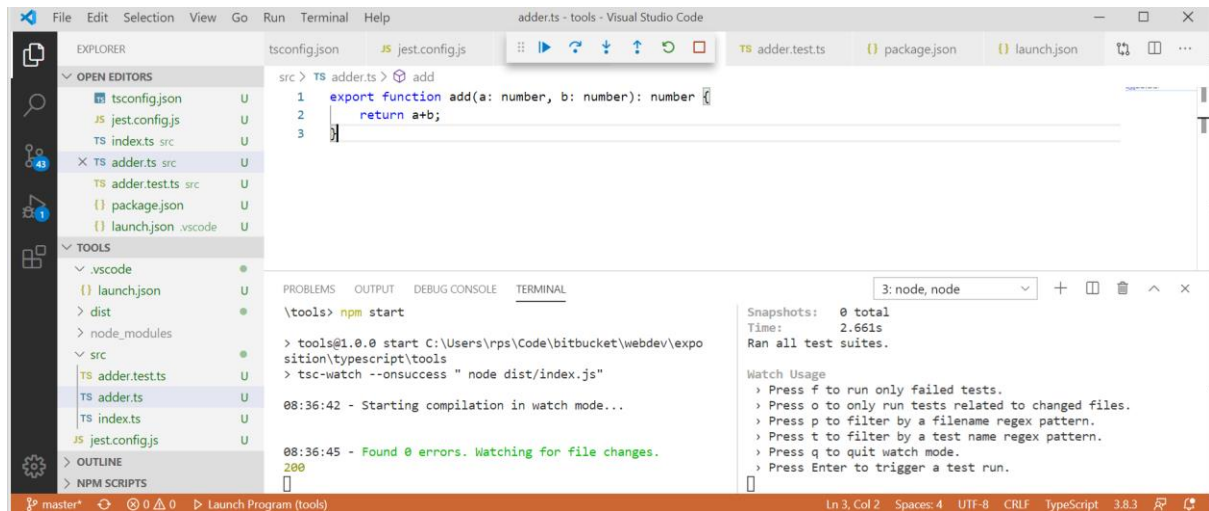
```
import {add} from "../adder";

test("do a test", () => {
  let result = add(10,5);
  expect(result).toBe(15);
})
```

Putting it together

Often it is useful to have two terminal windows: one with a file watcher compiling and running our application and one running the tests.

```
npm start
npm test
```



Specified Single File

We need to make sure we have the typescript compiler installed.

```
npm init -yes
npm install --save-dev typescript
```

We need to create a typescript compiler configuration file

Listing 2 tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2018", ❷
    "outDir": "./dist",
    "rootDir": "./src",
    "noEmitOnError": true,
    "sourceMap": true,
    "module": "commonjs" ❶
  }
}
```

- | | |
|------------------------|---|
| ❶ module format | Some environments such as node do not support ES2015 modules so specifying <code>commonjs</code> tells the compiler to generate older module code |
| ❷ target | The version of JavaScript to target |

COMPILE

When we run `tsc` from the command line with no arguments it will compile TypeScript source files in the `rootDir` to JavaScript files in the `outDir`

```
npx tsc
```

RUN SPECIFIED SINGLE FILE

We run JavaScript and not TypeScript, so the command is then

```
node dist/hello.js
```

RUN SPECIFIED SINGLE FILE WITH WATCH

To run typescript in watch mode we need an extra package called `tsc-watch`

```
npm install --save-dev tsc-watch
```

We then need to add a line to the scripts section in our `package.json`

```
"scripts": {  
  "test": "npx jest --watchAll",  
  "start": "tsc-watch --onSuccess \" node dist/hello.js\"",  
},
```

Selected File

RUN/DEBUG SPECIFIED FILE NO WATCH

Setup the `launch.json` with a configuration as follows.

```
{  
  "version": "0.2.0",  
  "configurations": [  
    {  
      "type": "node",  
      "request": "launch",  
      "name": "Run/Debug Open File",  
      "skipFiles": [  
        "<node_internals>/**"  
      ],  
      "program": "${file}",  
      "preLaunchTask": "tsc: build - tsconfig.json",  
      "outFiles": [  
        "${workspaceFolder}/**/*.js"  
      ]  
    }  
  ]  
}
```

You can then run/debug the current file using `Ctrl-F5` or `F5`

Jest

RUN ALL TESTS NO WATCH

To use jest with typescript we need the following

```
npm install --save-dev jest
npm install --save-dev @types/jest
npm install --save-dev @babel/preset-typescript
```

We also need a file called `babel.config.js`

```
module.exports = {
  presets: [
    ['@babel/preset-env', {targets: {node: 'current'}}],
    + '@babel/preset-typescript',
  ],
};
```

Finally, we run the tests as follows in the terminal

```
npx jest
```

RUN ALL TESTS WITH WATCH

```
npx jest --watchAll
```

RUN SINGLE FILE TEST NO WATCH

Run the test in `hello2.test.ts` Note we miss off the `.ts` from the filename

```
npx jest hello2.test
```

RUN SINGLE FILE TEST WATCH

```
npx jest hello2.test--watch
```

DEBUG/RUN SINGLE TEST FILE NO WATCH

Add the following configuration to `launch.json`

```
{
  "name": "Run/Debug Open Test",
  "type": "node",
  "request": "launch",
  "runtimeExecutable": "${workspaceRoot}/node_modules/.bin/jest
.cmd",
  "args": [
    "--runInBand",
    "--watchAll=false",
    "${fileBasenameNoExtension}"
  ],
  "cwd" : "${workspaceFolder}",
  "protocol": "inspector",
  "console": "integratedTerminal",
  "internalConsoleOptions": "neverOpen"
}
```

DEBUG/RUN SINGLE TEST FILE WITH WATCH

Add the following configuration to `launch.json`

```
{
  "name": "Run/Debug Open Test",
  "type": "node",
  "request": "launch",
  "runtimeExecutable": "${workspaceRoot}/node_modules/.bin/jest
.cmd",
  "args": [
    "--runInBand",
    "--watchAll=true",
    "${fileBasenameNoExtension}"
  ],
  "cwd" : "${workspaceFolder}",
  "protocol": "inspector",
  "console": "integratedTerminal",
  "internalConsoleOptions": "neverOpen"
}
```

CODE COVERAGE

```
npx jest --coverage
```

React

The following sections assume the project was setup using

```
npx create-react-app my-app --template typescript
```

RUN ALL TESTS NO WATCH

To use jest with typescript we need the following

```
npm install --save-dev jest
npm install --save-dev @types/jest
npm install --save-dev @babel/preset-typescript
```

We also need a file called `babel.config.js`

```
module.exports = {
  presets: [
    ['@babel/preset-env', {targets: {node: 'current'}}],
    + '@babel/preset-typescript',
  ],
};
```

Finally, we run the tests as follows in the terminal

```
npx jest
```

RUN ALL TESTS WITH WATCH

```
npx jest --watchAll
```

RUN SINGLE FILE TEST NO WATCH

Run the test in `hello2.test.ts` Note we miss off the `.ts` from the filename

```
npx jest hello2.test
```

RUN SINGLE FILE TEST WATCH

```
npx jest hello2.test--watch
```

DEBUG/RUN SINGLE TEST FILE NO WATCH

Add the following configuration to `launch.json`

```
{
  "name": "Run/Debug Open Test",
  "type": "node",
  "request": "launch",
  "runtimeExecutable": "${workspaceRoot}/node_modules/.bin/jest
.cmd",
  "args": [
    "--runInBand",
    "--watchAll=false",
    "${fileBasenameNoExtension}"
  ],
  "cwd" : "${workspaceFolder}",
  "protocol": "inspector",
  "console": "integratedTerminal",
  "internalConsoleOptions": "neverOpen"
}
```

DEBUG/RUN SINGLE TEST FILE WITH WATCH

Add the following configuration to `launch.json`

```
{
  "name": "Run/Debug Open Test",
  "type": "node",
  "request": "launch",
  "runtimeExecutable": "${workspaceRoot}/node_modules/.bin/jest
.cmd",
  "args": [
    "--runInBand",
    "--watchAll=true",
    "${fileBasenameNoExtension}"
  ],
  "cwd" : "${workspaceFolder}",
  "protocol": "inspector",
  "console": "integratedTerminal",
  "internalConsoleOptions": "neverOpen"
}
```

CODE COVERAGE

```
npx jest --coverage
```

React

This section assumes react with typescript was installed with

```
npx create-react-app my-app --template typescript
```

RUN ALL TESTS WITH WATCH

Simply run the following from the VSCode terminal

```
npm run tests
```

RUN SINGLE TEST WITH WATCH

If the test file is called maths.test.js we enter the command on the terminal

```
npm run test maths.test
```

DEBUG SINGLE TEST