Definition File Locations

In this lesson, you will explore where to find definition files

WE'LL COVER THE FOLLOWING

- Getting definition files directly from the JavaScript library
- Generating a definition file
- The case where no definition file is available

You can get definition files from two different places.

Getting definition files directly from the JavaScript library

The first one, which is getting popular, is directly from the JavaScript library when installing through NPM. If you look at the downloaded package, you might see index.d.ts in the folder of the library. If that is the case, it means that the author of the JavaScript library has provided the definition file. Having the library include the definition is the best case because the definition file is always in sync and up-to-date with the code.

The second way to get a definition file is to use NPM with `@types/X` where `X` is the name of the library you want the definition file from. The NPM package is not automatically valid for every library, but thousands and thousands of libraries got a package just for the type. The source code is from GitHub under the repository “DefinitelyTyped,” and it contains a collection of definition files across all libraries that contributors spend their time on, to keep the definition files up-to-date. Definition files are installed under the node_modules folder, under the folder `@types`, inside the folder of the library requested. The TypeScript transpiler knows to look in this folder. The location can also be changed in the compiler configuration file.

Generating a definition file

An author of a library in JavaScript that wants types for TypeScript consumers need to write a definition file. There is no magic to autogenerate the definition file, and it can involve quite a considerable amount of time to set up a file, depending on how big the library is. If you wrote your code in TypeScript, it's possible to produce the definition file directly when compiling by setting the `declaration` option to `true`. By doing so, you'll end up having a single definition file for your project all typed for you. Using the compiler is the quickest way. Since TypeScript knows the type of your project during compilation, all that information can be used to produce the definition file. The produced definition file can be copied and packaged with the JavaScript in the published package.

The case where no definition file is available

If there is no definition file available and you have no desire to create one, you need to create a fake definition file. This fake definition file lets you use the imported JavaScript file but won't give you any types. TypeScript needs to know about this file. The configuration file needs to have an entry to the location of the folder where the fake definition file resides. The following example gets the library's `pad` with NPM. Let's pretend that there is no definition file. The first step is to create a folder named `pad` in the @types folder, and then a file named `index.d.ts` with the following code.

```
declare module "pad";
```

The code can import the library and execute it. No Intellisense or type protection is provided since the definition file is defining only that `pad` is a module.

```
import * as pad from "pad";
console.log(pad("Test", 25, '+'));
```

Finally, TypeScript must know the location of the definition files. The configuration file lets you include the path for a definition file like any other TypeScript file. Moreover, TypeScript will check for the `@types` folder.

```
"include": [
  "src/**/*.ts",
  "@types/**/*.d.ts"
]
```