Exercise - Access external type libraries

5 minutes

Almost every project takes advantage of third-party libraries. TypeScript allows you to import libraries much in the same way you import modules you have created. However, unlike your modules the JavaScript library may not have type definitions.

Importing libraries

In JavaScript, you use external libraries in your code by using the `requires` statement. In TypeScript, you gain access to them by using the `import` statement. After importing a library and its type definition, you can use it in your code and gain the benefits of Intellisense and type checking.

Type libraries

Static typing is a primary reason to use TypeScript. External type libraries are available for almost all common libraries, providing this information for libraries that don't contain it (such as those written in JavaScript). The TypeScript compiler can raise an error message if you attempt to use a library that doesn't have type definitions. You'll also notice that Intellisense is not available when you lack these definitions.

While some JavaScript libraries will have type definitions, you will discover many do not. The open-source project DefinitelyTyped is a repository of TypeScript type definitions for many popular JavaScript libraries. You install type definitions by using the **@types** prefix.

Because the type definitions are only used by TypeScript during design-time, they are **not** required to be part of the published project. As a result, you can install them as devDependencies .

Bash

```
npm install --save-dev @types/<library-name>
```

> **Note**
>
> You must use an IDE, such as Visual Studio Code, to implement external type libraries. It is not possible to do this in the TypeScript Playground. Before completing the exercise, see the Lab setup section later in this module for more information about setting up a development environment in Visual Studio Code.

Exercise

In this exercise, you'll install and implement a type library called dotenv. This commonly used library loads environment variables from a **.env** file into `process.env`, enabling you to store configuration details in separate from your code and access them. You can use **dotenv** to manage things like connection strings and other values which may need to change depending on where your code is executing.

1. Open a new workspace in Visual Studio Code.
2. Add a new file called **Main.ts**.
3. From the terminal, generate a new **tsconfig.json** file with default configuration settings.

   Bash

   ```
   tsc --init
   ```

4. Go to DefinitelyTyped and search for the **dotenv** type library. DefinitelyTyped will provide the name of the library to install and other implementation details.
5. From the terminal, use `npm` to install the **dotenv** type library in your project folder.

   Bash

   ```
   npm install dotenv
   ```

6. The **dotenv** type definition also requires you to install the **node** type definition. **node** provides access to `process.env` so you can access it from your code.

   Bash

```
npm install --save-dev @types/node @types/dotenv
```

7. Create a new file in the root directory of your project called **.env**. This file will contain environment-specific variables for the project.

8. In **.env**, add the variables on new lines in the form of `NAME=VALUE`. For this example, define three variables:

TypeScript

```
DB_HOST=localhost
WEB_HOST=staging.adventure-works.com
```

9. In **Main.ts**, import the **dotenv** type library.

TypeScript

```
import dotenv from 'dotenv';
```

10. Assign `dotenv.config()` to a variable. `config` reads your **.env** file, parses the contents, assigns it to `process.env`, and returns an object with a `parsed` key containing the loaded content or an `error` key if it failed.

TypeScript

```
const result = dotenv.config();
```

11. TypeScript can now provide Intellisense and type checking for the `config` object. If you type `result.`, you see that `result` has two properties: `error` and `parsed`. Add an error checking statement to verify that the `config` operation worked as expect.

TypeScript

```
if (result.error) {
    throw result.error;
}
```

12. Return the contents of the `parsed` property to the console.

```
TypeScript

console.log(result.parsed);  // Returns { DB_HOST: 'localhost', WEB_HOST:
'staging.adventure-works.com' }
```

13. Access the values contains in each key in `process.env` and return the value to the console.

```
TypeScript

console.log(process.env.DB_HOST);
console.log(process.env.WEB_HOST);
```

This exercise provides an example of using **dotenv**. See dotenv to learn more about using it in your code.

Exercise solution

To see the solution to this exercise, clone the repository by entering the following at the command prompt.

```
Bash

git clone https://github.com/MicrosoftDocs/mslearn-typescript
cd mslearn-typescript/code/module-07/module07-exercise-02-end
code .
```

Next unit: Lab - Export and import module components

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆