Jump to bottom

Clarify how to add a new declaration file #21344

Closed **mjomble** opened this issue on Jan 22, 2018 · 14 comments

Labels Needs More Info

mjomble commented on Jan 22, 2018 • edited

Search Terms: declaration, adding declaration, add new declaration, clarify

Type of issue: requesting a change to an error message in `tsc` to improve clarity

Problem description

When I compile a file using `noImplicitAny` and an npm module that's missing a declaration file, I see a suggestion like this:

> Try `npm install @types/xml-escape` if it exists or add a new declaration (.d.ts) file containing `declare module 'xml-escape';`

It is not clear from this message where I should add the new `.d.ts` file, what should the full name of the file be or whether this is even important.

Since these details are not mentioned, my first assumption was the latter - that all `*.d.ts` files in the project are scanned and merged together.
VS Code seems to be doing something like that - I created a new `declarations.d.ts` file in the same dir as the file that uses the `xml-escape` module and it "automagically" picked up the declaration.

However, I later discovered that `tsc` works differently as it was still not able to find the declaration and showed me the above suggestion.

- I need to create a directory in my project for local declarations of npm modules
- I need to refer to this directory from `tsconfig.json`, for example `"typeRoots": ["./src/types"]`
- Based on the above example, the declaration for the `xml-escape` module must be in `[project root]/src/types/xml-module/index.d.ts`

Proposed change

The "add a new declaration file" suggestion should contain all of this information or include a link to a straight-forward guide. The end user should be able to figure out a correct file path in a few minutes rather than hours.

I would submit a PR, but I don't know of a single page with clear information. Ideally, this page should be part of the official documentation.
The best I've found so far are these two separate pages:

- https://www.typescriptlang.org/docs/handbook/tsconfig-json.html#types-typeroots-and-types - contains some hints as to how `typeRoots` is used, but leaves many details unexplained (I can elaborate on which parts were unclear, if needed)
- https://www.typescriptlang.org/docs/handbook/declaration-files/templates/module-d-ts.html - mentions that a declaration file for the module 'super-greeter' should go in `super-greeter/index.d.ts`, but leaves out the part that the super-greeter directory should be a subdir of a typeRoot.

Is there a part of the documentation that I've missed?

👍 18

aluanhaddad commented on Jan 30, 2018 • edited

@mjomble your conclusions are not correct.

For example, I will often have such a file, say `ambient.d.ts` (but it can have any valid filename), that contains a few declarations like yours.

```
// ambient.d.ts
declare module 'xyz';
```

and so long as it is within the set of files included in compilation the context I will not receive any errors when compiling code that imports it using `tsc`.
If you are receiving errors, then you have likely specified the `files` or `include` options in a way that excludes it. By default it will be included unless excluded.
For example, the following works fine:

directory structure:

`src/app.ts`

files:

src/app.ts:

```
import xyz from 'xyz';
```

ambient.d.ts:

```
declare module 'xyz';
```

tsconfig.json:

```
{
  "compilerOptions": {
    "module": "commonjs"
  }
}
```

Therefore, while it might be good to hint that the declaration needs to be specified at global scope, it does not need to adhere to any of the three criteria you mention above.

👍 6 ♡ 2

mhegazy commented on Jan 30, 2018

> It is not clear from this message where I should add the new .d.ts file, what should the full name of the file be or whether this is even important.

Just like **@aluanhaddad** noted, you can put this in any .d.ts file in your project.

> that all *.d.ts files in the project are scanned and merged together.

That is correct. files have different scopes though. a module (i.e. a file with at least one top-level import ot export) has its own scope, and declarations inside this file are not visible outside it unless exported.

> However, I later discovered that tsc works differently as it was still not able to find the declaration and showed me the above suggestion.

VSCode uses tsserver, which is the same as tsc. So i would expect them to behave the same. if that is not the same it is a bug.

this is not necessary. please share a repro where the simple solution is not working.

mhegazy added the Needs More Info label on Jan 30, 2018

mjomble commented on Feb 1, 2018

Hmm, quite annoyingly I've forgotten the exact way I tested this with `tsc`.
My main use case was using `ts.createProgram` in a custom build script.
I've managed to reduce that to a minimal example:

foo.d.ts:

```
declare module 'xml-escape' {
  function escape(string: string): string;
  export = escape;
}
```

xml.ts:

```
import xmlEscape = require('xml-escape')
```

build.js:

```
const ts = require('typescript')
const options = { noImplicitAny: true }
const host = ts.createCompilerHost(options)
const fileNames = ['xml.ts']
const program = ts.createProgram(fileNames, options, host)
const diagnostics = ts.getPreEmitDiagnostics(program)
console.log(ts.formatDiagnostics(diagnostics, host))
```

Running `node build.js` gives a "Could not find a declaration file for module 'xml-escape'" error and the suggestion to add a `.d.ts` file.

There are two ways that I've discovered to fix this:

1. Move/rename `foo.d.ts` to `customTypeRoot/xml-escape/index.d.ts` and change `options` in `build.js` to `{ noImplicitAny: true, typeRoots: ['./customTypeRoot'] }` (or use a different name/location for the custom type root)
2. Explicitly list the declaration file as a root name by changing `fileNames` in `build.js` to `['foo.d.ts', 'xml.ts']`

I noticed that VS Code was able to find the declaration file automatically while `ts.createProgram` could not.

I then wanted to test how `tsc` behaved, but due to the custom build process and the fact that the project was not yet entirely converted from Flow to TypeScript, it wasn't just a matter of running `tsc` in the root dir without any parameters.

Eventually, I managed to run it so that it worked with a custom type root (despite the unfinished project conversion) and failed without it. This lead me to the conclusion that `tsc` behaved more like `ts.createProgram` than VS Code.

However, I no longer remember the exact way I ran it and am unable to reproduce that scenario.
With the new, simpler example it does seem like `tsc` is behaving like VS Code, as expected.

So my revised conclusion is that `ts.createProgram` behaves differently from `tsc` and VS Code / tsserver when processing declaration files.
And I'm not sure if that's a bug or expected behavior :D

mhegazy commented on Feb 1, 2018

> Running node build.js gives a "Could not find a declaration file for module 'xml-escape'" error and the suggestion to add a .d.ts file.

well.. it kinda make sense, you did not give it `foo.d.ts` as an input file.. so not sure why you expect it to find it.

> When I reported this issue, I was working with a more complex project where the build script reads compiler options from the same tsconfig.json file that tsc uses and sends it through ts.convertCompilerOptionsFromJson.

It could be a miss use of the API, or a missing host function implementation.

> So my revised conclusion is that ts.createProgram behaves differently from tsc and VS Code / tsserver when processing declaration files.

I do not think this is accurate.. i think VSCode behaves diffrentelly from your custom use of `ts.createProgram`.

mjomble commented on Feb 1, 2018

> well.. it kinda make sense, you did not give it `foo.d.ts` as an input file.. so not sure why you expect it to find it.

But after finding out that VS Code / tsserver and tsc do find declarations from random `d.ts` files by scanning all of them within the project, I thought that maybe `ts.createProgram` is supposed to do that as well.

> > So my revised conclusion is that ts.createProgram behaves differently from tsc and VS Code / tsserver when processing declaration files.
>
> I do not think this is accurate.. i think VSCode behaves diffrentelly from your custom use of ts.createProgram.

To clarify, what I meant was:

- ts.createProgram behaves in one way (`d.ts` files must be explicitly referenced)
- tsc and VS Code / tsserver behave in a different way from that (all `d.ts` files are implicitly included regardless of their name and location, no explicit reference needed)

This seems consistent with what you said: VS Code behaves differently from ts.createProgram, at least in this particular aspect.

mhegazy commented on Feb 1, 2018

> The same way I assumed that VS Code / tsserver and tsc shouldn't find a declaration in a random d.ts file in a random location within the project, unless I explicitly refer to it somehow.

a tsconfig.json implicitly imply including all files in the directory.

mjomble commented on Feb 1, 2018

Ahh, there's the catch. My custom build logic only reads the `compilerOptions` section of `tsconfig.json` but it doesn't do anything with the `files`, `include` or `exclude` sections.
So, basically, to match the behavior of `tsc` in a situation where `tsconfig.json` has neither `files` nor `include` specified, one should basically pass in a list of all code and declaration files as root names to `ts.createProgram`.
That clears up the different behavior question, thanks.

However, one issue from my original comment still remains - the exact behavior of `typeRoots` is not very clearly documented.
It will result in some declaration files getting included even if they're not listed as a root name for `ts.createProgram` or if `tsconfig.json` specifies a fileset that doesn't directly include them.
But the documentation doesn't seem to make it very clear how to determine the exact location for these declaration files.

> It will result in some declaration files getting included even if they're not listed as a root name for ts.createProgram or if tsconfig.json specifies a fileset that doesn't directly include them.

They will always be included.. all the packages in your typeRoots are always included..

> However, one issue from my original comment still remains - the exact behavior of typeRoots is not very clearly documented.

Please see http://www.typescriptlang.org/docs/handbook/tsconfig-json.html#types-typeroots-and-types. Feel free to ask for specific clarifications and we would be happy to include it in the docs.

mjomble commented on Feb 1, 2018

The documentation mentions "packages" several times, but doesn't say much about what a package is and what exactly is in it.

For example, let's say I have a typeRoot configured at `src/types` and I'm getting an error about a missing declaration file for the `xml-escape` package.
I try installing `@types/xml-escape` from npm, but this package does not exist.

From the "**@types**, typeRoots and types" section of the documentation, I can conclude that I should create a local package for `xml-escape` inside `src/types`, but it's not obvious how exactly to do that.

I would assume that I need to write the module declaration into a file, but where exactly should I put this file and what should be its name?
Based only on the documentation, without trial and error, I wouldn't know which one (if any) of these paths is correct:

```
src/types/packages.ts
src/types/packages.d.ts
src/types/xml-escape.ts
src/types/xml-escape.d.ts
src/types/xml-escape.package
src/types/xml-escape.package.ts
src/types/xml-escape.package.d.ts
src/types/package.xml-escape.ts
src/types/package.xml-escape.d.ts
src/types/packages/xml-escape.ts
src/types/packages/xml-escape.d.ts
src/types/packages/xml-escape/index.ts
src/types/packages/xml-escape/index.d.ts
src/types/@types/xml-escape.ts
src/types/@types/xml-escape.d.ts
src/types/@types/xml-escape/index.ts
src/types/@types/xml-escape/index.d.ts
src/types/xml-escape/index.ts
src/types/xml-escape/index.d.ts
src/types/xml-escape/package.ts
src/types/xml-escape/package.d.ts
```

mhegazy commented on Feb 1, 2018

A package is an npm-package-like folder. i.e. a folder with a file called `index.d.ts` or a folder with a package.json that has a types field.

mjomble commented on Feb 1, 2018

Yes, I know that know, but this was not obvious from the documentation when I tried to find this information for the first time. It might be a good idea to include this information in the documentation.

PS. I apologize if I sound too complain-y on this issue. TypeScript is an awesome project and seems like a much better fit for me than Flow in many ways. Thank you for making it free and open sourced! :)

👍 2

mhegazy pushed a commit to microsoft/TypeScript-Handbook that referenced this issue on Feb 1, 2018

`Add explanation of what a type package is` … Verified ✓ 8423f46

mhegazy commented on Feb 1, 2018

Added explanation about what a type package is in microsoft/TypeScript-Handbook@`8423f46`#diff-66e160cc21e4a125c72d8025e5fe4e99

👍 3

mjomble commented on Feb 1, 2018

Thanks!

mjomble closed this on Feb 1, 2018

wmhilton commented on May 8, 2018

Thanks **@aluanhaddad** for the detailed example. This had been bugging me for two days.

Assignees

No one assigned

Labels

Needs More Info

Projects

None yet

Milestone

No milestone

4 participants