Usage of Angular libraries published to npm

When you build your Angular application, take advantage of sophisticated first-party libraries, as well as rich ecosystem of third-party libraries. Angular Material is an example of a sophisticated first-party library. For links to the most popular libraries, see Angular Resources.

Install libraries

Libraries are published as npm packages, usually together with schematics that integrate them with the Angular CLI. To integrate reusable library code into an application, you need to install the package and import the provided functionality in the location you use it. For most published Angular libraries, use the `ng add <lib_name>` Angular CLI command.

The `ng add` Angular CLI command uses a package manager to install the library package and invokes schematics that are included in the package to other scaffolding within the project code. Examples of package managers include npm or yarn. Additional scaffolding within the project code includes import statements, fonts, and themes.

A published library typically provides a `README` file or other documentation on how to add that library to your application. For an example, see the Angular Material documentation.

Library typings

Typically, library packages include typings in `.d.ts` files; see examples in `node_modules/@angular/material`. If the package of your library does not include typings and your IDE complains, you might need to install the `@types/<lib_name>` package with the library.

For example, suppose you have a library named `d3`:

npm install d3 --save npm install @types/d3 --save-dev

Types defined in a `@types/` package for a library installed into the workspace are automatically added to the TypeScript configuration for the project that uses that library. TypeScript looks for types in the `node_modules/@types` directory by default, so you do not have to add each type package individually.

If a library does not have typings available at `@types/`, you may use it by manually adding typings for it. To do this:

1. Create a `typings.d.ts` file in your `src/` directory. This file is automatically included as global type definition.
2. Add the following code in `src/typings.d.ts`:

   declare module 'host' { export interface Host { protocol?: string; hostname?: string; pathname?: string; } export function parse(url: string, queryString?: string): Host; }
3. In the component or file that uses the library, add the following code:

   import * as host from 'host'; const parsedUrl = host.parse('https://angular.io'); console.log(parsedUrl.hostname);

Define more typings as needed.

Updating libraries

A library is able to be updated by the publisher, and also has individual dependencies which need to be kept current. To check for updates to your installed libraries, use the `ng update` Angular CLI command.

Use `ng update <lib_name>` Angular CLI command to update individual library versions. The Angular CLI checks the latest published release of the library, and if the latest version is newer than your installed version, downloads it and updates your `package.json` to match the latest version.

When you update Angular to a new version, you need to make sure that any libraries you are using are current. If libraries have interdependencies, you might have to update them in a particular order. See the Angular Update Guide for help.

Adding a library to the runtime global scope

If a legacy JavaScript library is not imported into an application, you may add it to the runtime global scope and load it as if it was added in a script tag. Configure the Angular CLI to do this at build time using the `scripts` and `styles` options of the build target in the `angular.json` workspace build configuration file.

For example, to use the Bootstrap 4 library

1. Install the library and the associated dependencies using the npm package manager:

   npm install jquery --save npm install popper.js --save npm install bootstrap --save

2. In the `angular.json` configuration file, add the associated script files to the `scripts` array:

   "scripts": ["node_modules/jquery/dist/jquery.slim.js", "node_modules/popper.js/dist/umd/popper.js", "node_modules/bootstrap/dist/js/bootstrap.js"],

3. Add the `bootstrap.css` CSS file to the `styles` array:

   "styles": ["node_modules/bootstrap/dist/css/bootstrap.css", "src/styles.css"],

4. Run or restart the `ng serve` Angular CLI command to see Bootstrap 4 work in your application.

Using runtime-global libraries inside your app

After you import a library using the "scripts" array, do **not** import it using an import statement in your TypeScript code. The following code snippet is an example import statement.

import * as $ from 'jquery';

If you import it using import statements, you have two different copies of the library: one imported as a global library, and one imported as a module. This is especially bad for libraries with plugins, like JQuery, because each copy includes different plugins.

Instead, run the `npm install @types/jquery` Angular CLI command to download typings for your library and then follow the library installation steps. This gives you access to the global variables exposed by that library.

Defining typings for runtime-global libraries

If the global library you need to use does not have global typings, you can declare them manually as `any` in `src/typings.d.ts` .

For example:

declare var libraryName: any;

Some scripts extend other libraries; for instance with JQuery plugins:

$('.test').myPlugin();

In this case, the installed `@types/jquery` does not include `myPlugin`, so you need to add an interface in `src/typings.d.ts`. For example:

interface JQuery { myPlugin(options?: any): any; }

If you do not add the interface for the script-defined extension, your IDE shows an error:

[TS][Error] Property 'myPlugin' does not exist on type 'JQuery'

@reviewed 2022-01-05