The Angular Library Series - Building and Packaging

Building, packaging, and actually using our Angular library in a separate application.

Todd Palmer Follow
Jun 13, 2018 · 7 min read

The National Library of Finland

Fan of Angular-In-Depth? Support us on Twitter!

In this second article in **The Angular Library Series** we will:

- Explore what happens when we build our library.
- Package our library using npm pack.
- Actually use our library in a separate application.

A Quick Review

In the previous article we created a workspace using the **Angular CLI**. Then we used the Angular CLI to generate a library called **example-ng6-lib**. This resulted in two projects in our workspace:

- A library project in projects/example-ng6-lib
- A main application project in src/app

We were able to import the library into our application and test using a new component that we added.

For your convenience I have created a GitHub repository at:
t-palmer/example-ng6-lib
with the completed code from the previous article.

Building Our Library

Let's begin this second leg of our journey through wonders of Angular libraries by added a script to simplify building. Modify the scripts in the root package.json to add a build_lib script:

```
"scripts": {
  ...
  "build_lib": "ng build example-ng6-lib",
  ...
},
```

To build our library we can now use: `npm run build_lib`
This creates an **example-ng6-lib directory** in our workspace's **dist** directory.

After building our library we can build our application using:
`ng build`

This creates an **example-ng6-lib-app directory** in our workspace's **dist** directory.

package.json Files

"What?! More package.json files?"

Yes, I know. After building the library, there are at least three of them so far in the workspace. These can all get rather confusing so let's just take a minute to understand them.

Root package.json

This is the **main package.json** file for our library workspace. We use this to list our dependencies for both the application and the library. Any package that is needed to run or build either the application or the project must be in here.

When we use **npm install** during our development, this is the package.json where the new package will be added.

Library project package.json

The **library project package.json** is in the **projects\example-ng6-lib** directory and tells **ng-packagr** what information goes into the distribution package.json that will be shipped with our library. There are three critical pieces of information here:

- **name**
  Obviously, this is the name of our library. In the future when someone actually imports a module from our library, this is the name that will appear in quotes in the `from` part, for example:
  `import { ExampleNg6LibModule } from 'example-ng6-lib';`
- **version**
  The version of our library will be important so our users know that they have the latest updates. Developers using packages from npm typical expect you to use semantic versioning.
- **dependencies**
  This contains only the dependencies necessary to run our library. Hence, you will see both **dependencies** and **peerDependencies** but not **devDependencies** in this one.

You also can and should add the typical npm things such as license, author, repository, etc. to this file.

Note that when you use **npm install**, those packages only get added to the root package.json and NOT this one here in the library project. Therefore, whenever you install a new package your library uses, you will want to add it here manually. In a future article I hope to discuss the difference between **dependencies** and **peerDependencies**.

Library distribution package.json

The **library distribution package.json** is generated by **ng-packagr** in the **dist\example-ng6-lib** directory when we build our library. It is the package.json that gets released with our library.

Developers who want to use our library will use npm install which will look at this package.json to determine what other dependencies need to be installed.

Since this **distribution package.json** is generated for us by **ng-packagr**, you should NOT edit it directly. To change the contents, update the **project package.json** in the projects\example-ng6-lib folder. **ng-packagr** looks at that one as uses it as a basis to generate the **distribution package.json**.

> *REMEMBER: Never directly modify the library distribution package.json.*

Packaging Our Library

Packaging our library means taking the generated distribution files and creating a single tgz file that we can share either manually or on npm.

Let's create a script in our **root package.json** to package our library using **npm pack**.

```
"scripts": {
  ...
  "npm_pack": "cd dist/example-ng6-lib && npm pack",
  ...
},
```

To package our library we can now use: `npm run npm_pack`
This changes to our **library's dist folder** and runs **npm pack** which creates a package file named something like: **example-ng6-lib-0.0.1.tgz**

Like any good developer, I am lazy about repetitive tasks. So I like to create a script that combines both the build_lib and npm_pack scripts. Add the following to the scripts object in package.json:

```
"package": "npm run build_lib && npm run npm_pack"
```

Looking at our **root package.json** the relevant scripts look like this:

```
"scripts": {
  ...
  "build_lib": "ng build example-ng6-lib",
  "npm_pack": "cd dist/example-ng6-lib && npm pack",
  "package": "npm run build_lib && npm run npm_pack"
},
```

Notice that running the **package script** calls our **build_lib script** and then our **npm_pack script**.

Make sure you have built and packaged your library using:

```
npm run package
```

Our package script does two things:

1. Builds our library to the directory:
   dist/example-ng6-lib
2. Uses **npm pack** to create our library as an npm package in that directory:
   example-ng6-lib-0.0.1.tgz

Note that even though it is a tgz file, you can't just zip up the dist directory in tgz format.

> *ALWAYS: Use npm pack to create the tgz file.*

Using the Library in a Separate Application

Now that we know how to build and package an Angular library, let's create another test workspace so we can see what it looks like for people to actually use the library in their own application.

Creating a test workspace

Before you try to create a new workspace, verify that you are not inside your example-ng6-lib workspace. You want to be in its parent directory. Then using the **Angular CLI** we create a new workspace as a sibling to our existing example-ng6-lib workspace:

```
ng new lib-tester
cd lib-tester
ng serve
```

Remember, if you need to support IE, see Angular and Internet Explorer.

When we point our browser at:
http://localhost:4200/
we see the Angular CLI default application which I won't bore you with by displaying another screen shot.

Installing the library

So we have our test application. We want to use the library in our application so we use **npm install** like this:

```
npm install ../example-ng6-lib/dist/example-ng6-lib/example-
ng6-lib-0.0.1.tgz
```

Open up the package.json in the lib-tester workspace and you should see example-ng6-lib added to your dependencies. Mine looks like this:

```
"dependencies": {
  "@angular/animations": "^6.0.3",
  "@angular/common": "^6.0.3",
  "@angular/compiler": "^6.0.3",
  "@angular/core": "^6.0.3",
  "@angular/forms": "^6.0.3",
  "@angular/http": "^6.0.3",
  "@angular/platform-browser": "^6.0.3",
  "@angular/platform-browser-dynamic": "^6.0.3",
  "@angular/router": "^6.0.3",
  "core-js": "^2.5.4",
  "example-ng6-lib": "file:../example-ng6-lib/dist/example-
ng6-lib/example-ng6-lib-0.0.1.tgz",
  "rxjs": "^6.0.0",
  "zone.js": "^0.8.26"
},
```

Also, if you look in the node_modules directory of our workspace, you will see an **example-ng6-lib** directory for the library.

WARNING: The first time I built a library way back in days of yore, I knew just enough about npm to be dangerous. So I thought, "Wait. I can just npm install the project dist directory instead of the package. Then, when I update my code, my test application will automatically pick up the changes." Only do this if you enjoy bashing your head against a wall as you struggle with strange errors about 3rd party imports. You have been warned.

> *ALWAYS: Install the library's .tgz package and NOT the directory.*

Importing the library module

In order to actually use a component from the library we need to add the library's module to our App Module.

To do this we make two changes in: **src\app\app.module.ts**

1. Import the **ExampleNg6LibModule**:
   `import { ExampleNg6LibModule } from 'example-ng6-lib';`
2. Add the **ExampleNg6LibModule** to the `imports` array in our **AppModule**:

Your **app.module.ts** file should look like this:

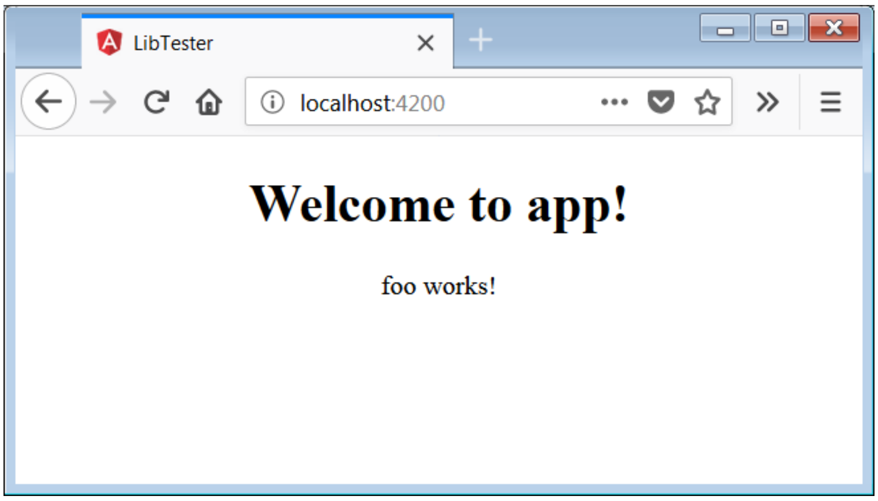
```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { ExampleNg6LibModule } from 'example-ng6-lib';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    ExampleNg6LibModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Actually using a component from our library

And now just like we did in the library's application we can use the **enl-foo** component. In the lib-tester application modify the html template for **AppComponent** and display the **FooComponent** from the library. After making the changes, **app.component.html** should look something like this:

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  <enl-foo></enl-foo>
</div>
```

Then when we point our browser at http://localhost:4200/ we can see that our lib-tester application is correctly displaying the component.



Welcome to app!

foo works!

And there you have it!

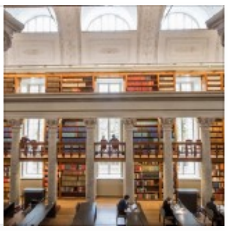
Looking Forward

In part 3: The Angular Library Series—Publishing we will tie up some loose ends about actually publishing your library to npm.

The Angular Library Series—Publishing

Publishing your Angular Library to npm

blog.angularindepth.com



. . .

3 reasons why you should follow Angular-In-Depth publication