The Angular Library Series—Publishing

Publishing your Angular Library to npm

Todd Palmer Follow
Aug 29, 2018 · 6 min read

Kansalliskirjasto: The National Library of Finland

So you have created and built your amazing library. Before we move on to dependencies I want to tie up some loose ends and show you how to share your new Angular Library with the world by publishing it on npm.

The Angular Library Series

This is the third article in my **Angular Library Series**. Before going forward with this article you should at least be familiar with the previous two articles in the series:

1. The Angular Library Series—Creating a Library with the Angular CLI
2. The Angular Library Series—Building and Packaging
3. The Angular Library Series—Publishing

An Example

By following the instructions in my previous articles in **The Angular Library Series**, I created a simple example Angular Library: **ng-example-library**

The source is available on my GitHub.

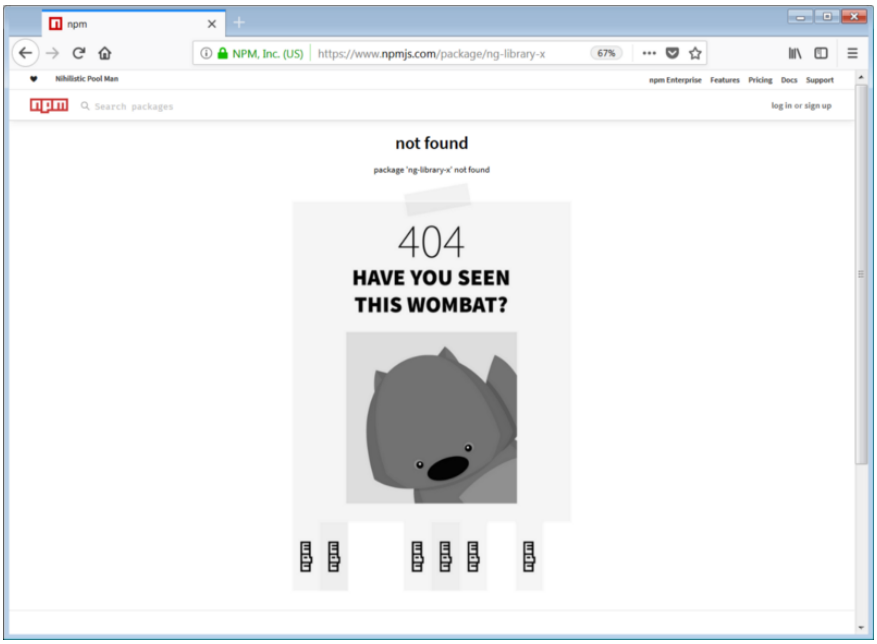
In this article I will show how I published it out to npm here.

Naming Your Angular Library

In order to publish your library to npm you will need to come up with a **unique name** for it. Before you actually create your library with Angular CLI, you will want to check npm first. This is because you don't really want to have to change the name of the library in your Angular Workspace once you have created it.

To check if the name you plan to use is already taken, you should try to go to the package's page on npm. For example, if your library's intended name is **ng-library-x**, you would check this URL: https://www.npmjs.com/package/ng-library-x

You want npm to show you an page like this:



Then you know that name isn't already being used on npm.

Yes, I know. You can search for packages on npm. But, this won't find deprecated packages. And, once a name has been used in the npm registry, it can't be used again by another user, even if it has been deprecated. So, let's add another one of our rules:

> *ALWAYS: Before creating your Workspace, check if your planned library name is unique by going to its intended npm URL.*

So Many package.json Files!

Just as a quick refresher: you remember that when we create an Angular library there are always at least three **package.json** files in our workspace:

- **Workspace Root package.json**
  This is the **main package.json** file in the root of our workspace.
- **Library Project package.json**
  This is in the **projects\ng-example-library** directory and tells **ng-packagr** what information goes into the Distribution package.json that will be shipped with our library.
- **Library Distribution package.json**
  This one is generated by **ng-packagr** in the **dist\ng-example-library** directory when we build our library. It is the package.json that gets published with our library.

If any of the above isn't perfectly clear to you, please review the **package.json** section in my previous article: Building and Packaging.

Specify Your Library's Version

Let's take a look at our **Library Project package.json** file. Mine looks like this:

```
{
  "name": "ng-example-library",
  "version": "1.0.0",
  "peerDependencies": {
    "@angular/common": "^6.0.0-rc.0 || ^6.0.0",
    "@angular/core": "^6.0.0-rc.0 || ^6.0.0"
  }
}
```

We are interested in the name and version entries. For now, you can ignore the `peerDependencies`, we will discuss them in the next article.

Note that npm only allows you to publish a specific name and version combination once. Hence, you will need to change the version each time you want to publish updates to your library on npm. In general you should use Semantic Versioning (SemVer) unless you have a really good reason not to.

> *PREFERABLY: Update your Library's version according to SemVer.*

Now when you build your library, you will see the version updated in the **Distribution package.json** in your library folder in the **dist** folder. Remember, this is the one that actually gets packaged with your library and is visible to the public. After building the library, my **Distribution package.json** looks like this:

```
{
  "name": "ng-example-library",
  "version": "1.0.0",
  "peerDependencies": {
    "@angular/common": "^6.0.0-rc.0 || ^6.0.0",
    "@angular/core": "^6.0.0-rc.0 || ^6.0.0"
  },
  "main": "bundles/ng-example-library.umd.js",
  "module": "fesm5/ng-example-library.js",
  "es2015": "fesm2015/ng-example-library.js",
  "esm5": "esm5/ng-example-library.js",
```

```
  "esm2015": "esm2015/ng-example-library.js",
  "fesm5": "fesm5/ng-example-library.js",
  "fesm2015": "fesm2015/ng-example-library.js",
  "typings": "ng-example-library.d.ts",
  "metadata": "ng-example-library.metadata.json",
  "sideEffects": false,
  "dependencies": {
    "tslib": "^1.9.0"
  }
}
```

Read-me and License Files

When you publish a package to npm, it looks for a **README** file in the root of the package. If it finds one, it uses that **README** as the front page for your package on npm.

Remember in my previous article Building and Packaging we created an npm script in our **Workspace package.json** file called `package`. This script builds and then packs our library. However, before packing we want to copy our **README.md file** and our **LICENSE file** to the **dist package**.

So, I created an **npm script** called **copy-files** that copies those files before I actually do the pack. These scripts are for my own Windows based deployment. So, your mileage may vary on other platforms.

```
"scripts": {
  ...
  "build_lib": "ng build ng-example-library",
  "copy-license": "copy .\\LICENSE .\\dist\\ng-example-
library",
  "copy-readme": "copy .\\README.md .\\dist\\ng-example-
library",
  "copy-files": "npm run copy-license && npm run copy-
readme",
  "npm_pack": "cd dist/ng-example-library && npm pack",
  "package": "npm run build_lib && npm run copy-files && npm
run npm_pack",
  ...
},
```

Notice that my **package** script is now doing three things:

1. **build_lib**
   Build the Angular Library.

2. **copy-files**
   Copy both **README.md** and **LICENSE** to the **dist\ng-example-library** folder.

3. **npm_pack**
   Package up the **dist\ng-example-library** folder into a **tgz** file.

More Information in package.json

If we were to publish our package now and then view it on npm, we would see **none** for the license. That's because even though we have a **LICENSE file** in our package, we haven't mentioned anything about licence in our package.json.

As a matter of fact, there is a bunch stuff that we can add to our library's package.json. npm provides good documentation in The specifics of npm's package.json handling. So, I won't repeat it here. But, I will highlight a couple of items that I think are important when publishing a library.

Remember, we never directly modify our **Distribution package.json**. So if you want to add information, you need to add it in you **Library's Project package.json**.

I am going to add the following information:

- **License**
  Reference the LICENSE file.
- **Repository**
  Point to the GitHub repository.
- **Description**
- **Key words**
- **Home page**
  Point to this article.

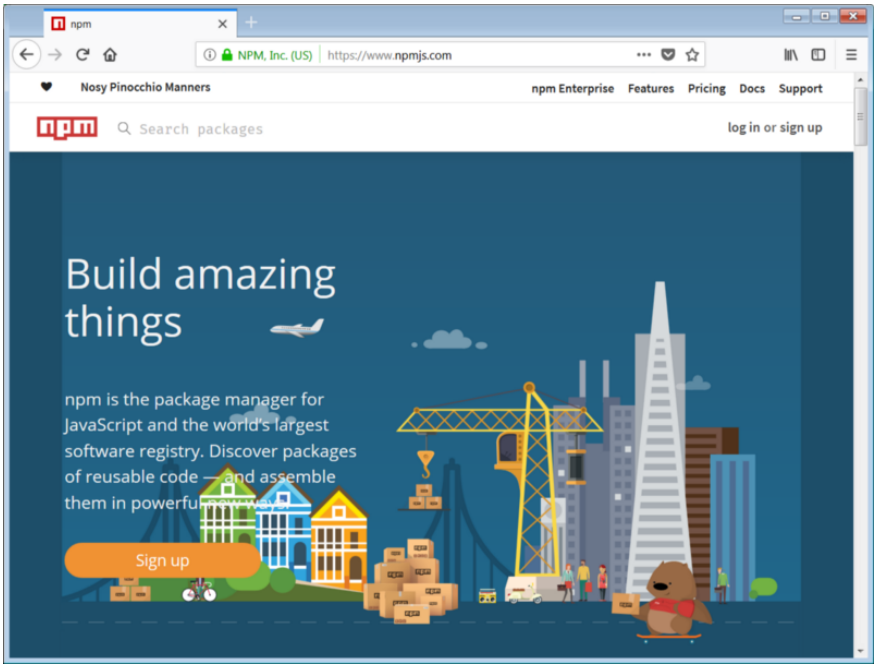
So now my library's **Project package.json** looks like this:

```
{
  "name": "ng-example-library",
  "version": "1.2.0",
  "description": "This is a simple example Angular Library
```

```
published to npm.",
  "keywords" :["Angular","Library"],
  "license" : "SEE LICENSE IN LICENSE",
  "repository": {
    "type" : "git",
    "url" : "https://github.com/t-palmer/ng-example-library"
  },
  "homepage" :"https://medium.com/@palmer_todd/the-angular-
library-series-publishing-ce24bb673275",
  "peerDependencies": {
    "@angular/common": "^6.0.0-rc.0 || ^6.0.0",
    "@angular/core": "^6.0.0-rc.0 || ^6.0.0"
  }
}
```

Sign Up for npm

You can freely install packages from npm to your hearts content. But, if you want to actually **publish a package** you need to **sign up for an account**. To sign up you will need to provide your full name, a user name and password, and a valid email address.



After signing up, make sure you check your inbox for the email verification.

Log in to npm

npm provides excellent instructions about how to actually publish your package. You will want to take a look at them. However, for your convenience I have added the critical steps here.

Before publishing you need to log in to npm using:

```
npm login
```

This will prompt you for your npm credentials and email. You can then verify that you are logged in using:

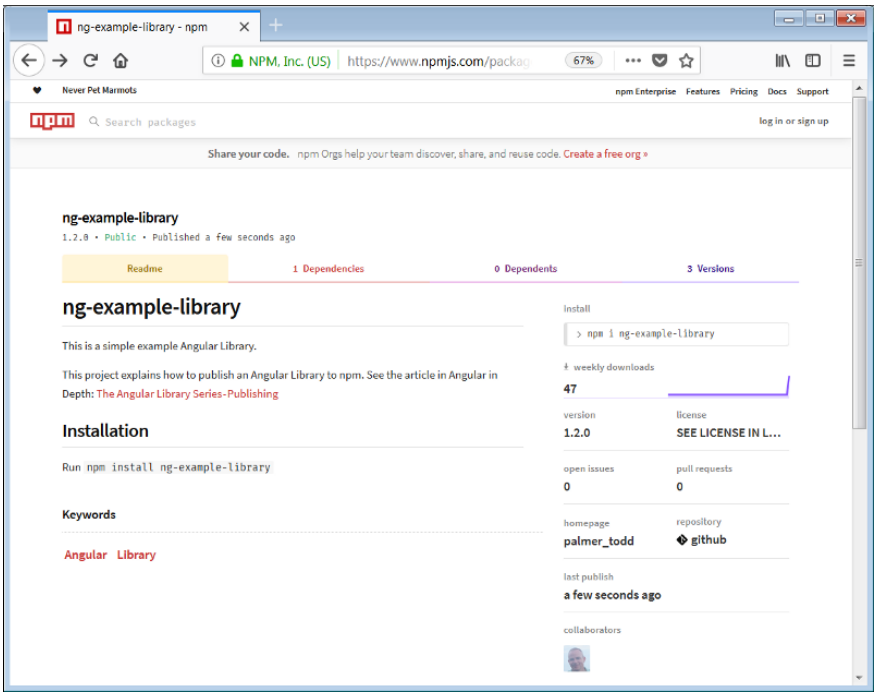
```
npm whoami
```

Publishing

Finally, we are ready to publish our library. Remember, we already used npm pack so we can just publish our **.tgz** file. For my example I used:

```
npm publish ./dist/ng-example-library/ng-example-library-
1.2.0.tgz
```

Then we can see the published package on npm at this URL: https://www.npmjs.com/package/ng-example-library



Looking Ahead

To get an understanding of how to handle dependencies for your library please take a look at my article: