NPM PACKAGES

Details of the recommended npm packages and the different kinds of package dependencies

Angular applications and Angular itself depend upon features and functionality provided by a variety of third-party packages (including Angular itself). These packages are maintained and installed with the Node Package Manager (npm).

> Node.js and npm are essential to Angular 2 development.
>
> Get it now if it's not already installed on your machine
>
> **Verify that you are running at least node `v4.x.x` and npm `3.x.x`** by running `node -v` and `npm -v` in a terminal/console window. Older versions produce errors.
>
> We recommend nvm for managing multiple versions of node and npm.

We recommend a comprehensive starter-set of packages as specified in the `dependencies` and `devDependencies` sections of the QuickStart package.json file:

package.json (dependencies)

```
{
  "dependencies": {
```

```
      "@angular/common": "2.0.0-rc.4",
      "@angular/compiler": "2.0.0-rc.4",
      "@angular/core": "2.0.0-rc.4",
      "@angular/forms": "0.2.0",
      "@angular/http": "2.0.0-rc.4",
      "@angular/platform-browser": "2.0.0-rc.4",
      "@angular/platform-browser-dynamic":
  "2.0.0-rc.4",
      "@angular/router": "3.0.0-beta.1",
      "@angular/router-deprecated": "2.0.0-rc.2",
      "@angular/upgrade": "2.0.0-rc.4",
      "systemjs": "0.19.27",
      "core-js": "^2.4.0",
      "reflect-metadata": "^0.1.3",
      "rxjs": "5.0.0-beta.6",
      "zone.js": "^0.6.12",
      "angular2-in-memory-web-api": "0.0.14",
      "bootstrap": "^3.3.6"
    },
    "devDependencies": {
      "concurrently": "^2.0.0",
      "lite-server": "^2.2.0",
      "typescript": "^1.8.10",
      "typings": "^1.0.4"
    }
  }
```

There are other possible package choices and you're likely to add and subtract to meet your application needs. We're recommending *this particular set* because (a) we know they work well together and (b) they include everything we'll need to build and run the sample applications in this documentation series.

> *Almost* everything. A cookbook or guide chapter may require an additional library such as *jQuery*.

This is far more than we need for QuickStart. Indeed, it's more than we need for most applications. There is no harm in

installing more than we need. We only serve to the client those packages that the application actually requests.

In this chapter we explain what each package does and why we include it. Feel free to make substitutions later to suit your tastes and experience.

dependencies and *devDependencies*

The `package.json` distinguishes between two sets of packages, dependencies and devDependencies.

The packages listed under *dependencies* are essential to *running* the application. The *devDependencies* are only necessary to *develop* the application. They can be excluded from production installations as in this example:

```
npm install my-application --production
```

dependencies

There are three package categories in the `dependencies` section of the application `package.json`:

- ***Features*** - Feature packages provide our application with framework and utility capabilities.
- ***Polyfills*** - Polyfills plug gaps in the browser's JavaScript implementation.
- ***Other*** - Other libraries that support the application such as `bootstrap` for HTML widgets and styling.

Feature Packages

@angular/core - Critical runtime parts of the framework needed by every application. Includes all metadata decorators, `Component`, `Directive`, dependency injection, and the component lifecycle hooks.

@angular/common - The commonly needed services, pipes and directives provided by the Angular team.

@angular/compiler - Angular's *Template Compiler*. It understand templates and can convert them to code that makes the app run and render. Developers typically don't interact with the compiler directly. They use it indirectly via `platform-browser-dynamic` or the offline template compiler.

@angular/platform-browser - Everything DOM and browser related, especially the pieces that help render into DOM. This package also includes the bootstrapStatic method for bootstrapping applications for production builds that pre-compile templates offline.

@angular/platform-browser-dynamic - Providers and a bootstrap method for applications that compile templates on the client. don't use offline compilation. We use this package for boostrapping during development and for boostrapping plunker samples

@angular/http - Angular's http client

@angular/router - Component router.

@angular/upgrade - Set of utilities for upgrading Angular 1 applications.

system.js - A dynamic module loader compatible with the ES2015 module specification. There are other viable choices including the well-regarded webpack. SystemJS happens to be the one we use in the documentation samples. It works.

Our applications are likely to require additional packages that provide HTML controls, themes, data access, and various utilities.

Polyfill Packages

Angular requires certain polyfills in the application

environment. We install these polyfills with very specific npm packages that Angular lists in the *peerDependencies* section of its `package.json`.

We must list these packages in the `dependencies` section of our own `package.json`.

> See "Why peerDependencies?" below for background on this requirement.

core-js - monkey patches the global context (window) with essential features of ES2015 (ES6). Developers may substitute an alternative polyfill that provides the same core APIs. This dependency should go away once these APIs are implemented by all supported ever-green browsers.

reflect-metadata - a dependency shared between Angular and the ***TypeScript compiler***. Developers should be able to update a TypeScript package without upgrading Angular, which is why this is a dependency of the application and not a dependency of Angular.

rxjs - a polyfill for the Observables specification currently before the TC39 committee that determines standards for the JavaScript language. Developers should be able to pick a preferred version of *rxjs* (within a compatible version range) without waiting for Angular updates.

zone.js - a polyfill for the Zone specification currently before the TC39 committee that determines standards for the JavaScript language. Developers should be able to pick a preferred version of *zone.js* to use (within a compatible version range) without waiting for Angular updates.

Other helper libraries

angular2-in-memory-web-api - An Angular-supported library that simulates a remote server's web api without requiring an

actual server or real http calls. Good for demos, documentation samples, and early stage development (before we even have a server). Learn about it in the Http Client chapter.

bootstrap - bootstrap is a popular HTML and CSS framework for designing responsive web apps. Some of the documentation samples improve their appearance with *bootstrap*.

devDependencies

The packages listed in the *devDependencies* section of the `package.json` help us develop the application. They do not have to be deployed with the production application although there is rarely harm in doing so.

concurrently - a utility to run multiple *npm* commands concurrently on OS/X, Windows, and Linux operating systems.

lite-server - a light-weight, static file server, written and maintained by John Papa with excellent support for Angular apps that use routing.

typescript - the TypeScript language server including the *tsc* TypeScript compiler.

typings - a manager for TypeScript definition files. Learn more about it in the TypeScript Configuration chapter.

This section likely grows as we add more tools, testing, and build support. The QuickStart set is sufficient for developing the documentation sample applications.

Why *peerDependencies*?

We don't have a *peerDependencies* section in the QuickStart `package.json`. But Angular itself has a *peerDependencies* section in *its* package.json and that has important consequences for our application.

It explains why we load the polyfill *dependency* packages in the QuickStart `package.json`, and why we'll need those packages in our own applications.

Let's briefly explain what peer dependencies are about.

As we know, packages depend on other packages. For example, our application depends upon the Angular package.

Two packages, 'A' and 'B', could depend on the same third package 'C'. 'A' and 'B' might both list 'C' among their *dependencies*.

What if 'A' and 'B' depend on different versions of 'C' ('C1' and 'C2'). The npm package system supports that! It installs 'C1' in the `node_modules` folder for 'A' and 'C2' in the `node_modules` folder for 'B'. Now 'A' and 'B' have their own copies of 'C' and they run without interferring. This is great.

But there is a problem. Package 'A' may require the presence of 'C1' without actually calling upon it directly. 'A' may only work if *everyone is using 'C1'*. It falls down if any part of the application relies on 'C2'.

The solution is for 'A' to declare that 'C1' is a *peer dependency*.

The difference between a `dependency` and a `peerDependency` is roughly this:

> A **dependency** says, "I need this thing directly available to *me*."
>
> A **peerDependency** says, "if you want to use me, you need this thing available to *you*."

Angular finds itself in this situation. Accordingly, the Angular `package.json` specifies several *peer dependency* packages, each pinned to a particular version of a third-party package.

We must install Angular's *peerDependencies* ourselves

When *npm* installs packages listed in *our* `dependencies`

section, it also installs the packages listed within *their* packages `dependencies` sections. The process is recursive.

But as of version 3, *npm* does *not* install packages listed in *peerDependencies* sections.

That means when our application installs Angular, ***npm* will not automatically install the packages listed in Angular's *peerDependencies* section**.

Fortunately, *npm* warns us (a) when any *peer dependencies* are missing or (b) when the application or any its other dependencies installs a different version of a *peer dependency*.

These warnings are a critical guard against accidental failures due to version mismatches. They leave us in control of package and version resolution.

It is our responsibility to list all *peer dependency* packages **among our own *devDependencies***.

> ### The future of *peerDependencies*
>
> The Angular polyfill dependencies should be just a suggestion or a hint to developers so that they know what Angular expects. They should not be hard requirements as they are today. We don't have a way to make them optional today.
>
> There is a npm feature request for "optional peerDependencies" which would allow us to model this relationship better. Once implemented, Angular will switch from *peerDependencies* to *optionalPeerDependencies* for all polyfills.