Generating code using schematics

Contents ›

A schematic is a template-based code generator that supports complex logic. It is a set of instructions for transforming a software project by generating or modifying code. Schematics are packaged into collections and installed with npm.

The schematic collection can be a powerful tool for creating, modifying, and maintaining any software project, but is particularly useful for customizing Angular projects to suit the particular needs of your own organization. You might use schematics, for example, to generate commonly-used UI patterns or specific components, using predefined templates or layouts. You can use schematics to enforce architectural rules and conventions, making your projects consistent and inter-operative.

Schematics for the Angular CLI

Schematics are part of the Angular ecosystem. The Angular CLI uses schematics to apply transforms to a web-app project. You can modify these schematics, and define new ones to do things like update your code to fix breaking changes in a dependency, for example, or to add a new configuration option or framework to an existing project.

Schematics that are included in the `@schematics/angular` collection are run by default by the commands `ng generate` and `ng add`. The package contains named schematics that configure the options that are available to the CLI for `ng generate` sub-commands, such as `ng generate component` and `ng generate service`. The subcommands for `ng generate` are shorthand for the corresponding schematic. You can specify a particular schematic (or collection of schematics) to generate, using the long form:

```
ng generate my-schematic-collection:my-schematic-name
```

or

```
ng generate my-schematic-name --collection collection-name
```

Configuring CLI schematics

A JSON schema associated with a schematic tells the Angular CLI what options are available to commands and subcommands, and determines the defaults. These defaults can be overridden by providing a different value for an

option on the command line. See Workspace Configuration for information about how you can change the generation option defaults for your workspace.

The JSON schemas for the default schematics used by the CLI to generate projects and parts of projects are collected in the package `@schematics/angular`. The schema describes the options available to the CLI for each of the `ng generate` sub-commands, as shown in the `--help` output.

Developing schematics for libraries

As a library developer, you can create your own collections of custom schematics to integrate your library with the Angular CLI.

- An *add schematic* allows developers to install your library in an Angular workspace using `ng add`.
- *Generation schematics* can tell the `ng generate` subcommands how to modify projects, add configurations and scripts, and scaffold artifacts that are defined in your library.
- An *update schematic* can tell the `ng update` command how to update your library's dependencies and adjust for breaking changes when you release a new version.

For more details of what these look like and how to create them, see:

- Authoring Schematics
- Schematics for Libraries

Add schematics

An add schematic is typically supplied with a library, so that the library can be added to an existing project with `ng add`. The `add` command uses your package manager to download new dependencies, and invokes an installation script that is implemented as a schematic.

For example, the `@angular/material` schematic tells the `add` command to install and set up Angular Material and theming, and register new starter components that can be created with `ng generate`. You can look at this one as an example and model for your own add schematic.

Partner and third party libraries also support the Angular CLI with add schematics. For example, `@ng-bootstrap/schematics` adds ng-bootstrap to an app, and `@clr/angular` installs and sets up Clarity from VMWare.

An add schematic can also update a project with configuration changes, add additional dependencies (such as polyfills), or scaffold package-specific initialization code. For example, the `@angular/pwa` schematic turns your application into a PWA by adding an app manifest and service worker, and the `@angular/elements` schematic adds the `document-register-element.js` polyfill and dependencies for Angular Elements.

Generation schematics

Generation schematics are instructions for the `ng generate` command. The documented sub-commands use the default Angular generation schematics, but you can specify a different schematic (in place of a sub-command) to generate an artifact defined in your library.

Angular Material, for example, supplies generation schematics for the UI components that it defines. The following command uses one of these schematics to render an Angular Material `<mat-table>` that is pre-configured with a datasource for sorting and pagination.

```
ng generate @angular/material:table
```

Update schematics

The `ng update` command can be used to update your workspace's library dependencies. If you supply no options or use the help option, the command examines your workspace and suggests libraries to update.

```
ng update
    We analyzed your package.json, there are some packages to update:

      Name                               Version                  Command to update
     --------------------------------------------------------------------------------
      @angular/cdk                       7.2.2 -> 7.3.1           ng update
@angular/cdk
      @angular/cli                       7.2.3 -> 7.3.0           ng update
@angular/cli
      @angular/core                      7.2.2 -> 7.2.3           ng update
@angular/core
      @angular/material                  7.2.2 -> 7.3.1           ng update
@angular/material
      rxjs                               6.3.3 -> 6.4.0           ng update rxjs


    There might be additional packages that are outdated.
    Run "ng update --all" to try to update all at the same time.
```

If you pass the command a set of libraries to update (or the `--all` flag), it updates those libraries, their peer dependencies, and the peer dependencies that depend on them.

> If there are inconsistencies (for example, if peer dependencies cannot be matched by a simple semver range), the command generates an error and does not change anything in the workspace.
>
> We recommend that you do not force an update of all dependencies by default. Try updating specific dependencies first.
>
> For more about how the `ng update` command works, see Update Command.

If you create a new version of your library that introduces potential breaking changes, you can provide an *update schematic* to enable the `ng update` command to automatically resolve any such changes in the project being updated.

For example, suppose you want to update the Angular Material library.

```
ng update @angular/material
```

This command updates both `@angular/material` and its dependency `@angular/cdk` in your workspace's `package.json`. If either package contains an update schematic that covers migration from the existing version to a new version, the command runs that schematic on your workspace.

Version 10.0.10-local+sha.84d1ba792b.