Workspace and project file structure

You develop applications in the context of an Angular workspace. A workspace contains the files for one or more projects. A project is the set of files that comprise a standalone application or a shareable library.

The Angular CLI `ng new` command creates a workspace.

ng new <my-project>
When you run this command, the CLI installs the necessary Angular npm packages and other dependencies in a new workspace, with a root-level application named *my-project*. The workspace root folder contains various support and configuration files, and a README file with generated descriptive text that you can customize.

By default, `ng new` creates an initial skeleton application at the root level of the workspace, along with its end-to-end tests. The skeleton is for a simple Welcome application that is ready to run and easy to modify. The root-level application has the same name as the workspace, and the source files reside in the `src/` subfolder of the workspace.

This default behavior is suitable for a typical "multi-repo" development style where each application resides in its own workspace. Beginners and intermediate users are encouraged to use `ng new` to create a separate workspace for each application.

Angular also supports workspaces with multiple projects. This type of development environment is suitable for advanced users who are developing shareable libraries, and for enterprises that use a "monorepo" development style, with a single repository and global configuration for all Angular projects.

To set up a monorepo workspace, you should skip the creating the root application. See Setting up for a multi-project workspace below.

Workspace configuration files

All projects within a workspace share a CLI configuration context. The top level of the workspace contains workspace-wide configuration files, configuration files for the root-level application, and subfolders for the root-level application source and test files.

WORKSPACE CONFIG FILES	PURPOSE
`.editorconfig`	Configuration for code editors. See EditorConfig.
`.gitignore`	Specifies intentionally untracked files that Git should ignore.
`README.md`	Introductory documentation for the root application.
`angular.json`	CLI configuration defaults for all projects in the workspace, including configuration options for build, serve, and test tools that the CLI uses, such as Karma, and Protractor. For details, see Angular Workspace Configuration.
`package.json`	Configures npm package dependencies that are available to all projects in the workspace. See npm documentation for the specific format and contents of this file.
`package-lock.json`	Provides version information for all packages installed into `node_modules` by the npm client. See npm documentation for details. If you use the yarn client, this file will be yarn.lock instead.

`src/`	Source files for the root-level application project.
`node_modules/`	Provides npm packages to the entire workspace. Workspace-wide `node_modules` dependencies are visible to all projects.
`tsconfig.json`	The base TypeScript configuration for projects in the workspace. All other configuration files inherit from this base file. For more information, see the Configuration inheritance with extends section of the TypeScript documentation.

Application project files

By default, the CLI command `ng new my-app` creates a workspace folder named "my-app" and generates a new application skeleton in a `src/` folder at the top level of the workspace. A newly generated application contains source files for a root module, with a root component and template.

When the workspace file structure is in place, you can use the `ng generate` command on the command line to add functionality and data to the application. This initial root-level application is the *default app* for CLI commands (unless you change the default after creating additional apps).

Besides using the CLI on the command line, you can also manipulate files directly in the application's source folder and configuration files.

For a single-application workspace, the `src/` subfolder of the workspace contains the source files (application logic, data, and assets) for the root application. For a multi-project workspace, additional projects in the `projects/` folder contain a `project-name/src/` subfolder with the same structure.

Application source files

Files at the top level of `src/` support testing and running your application. Subfolders contain the application source and application-specific configuration.

APP SUPPORT FILES	PURPOSE
`app/`	Contains the component files in which your application logic and data are defined. See details below.
`assets/`	Contains image and other asset files to be copied as-is when you build your application.
`environments/`	Contains build configuration options for particular target environments. By default there is an unnamed standard development environment and a production ("prod") environment. You can define additional target environment configurations.
`favicon.ico`	An icon to use for this application in the bookmark bar.
`index.html`	The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any `<script>` or `<link>` tags here manually.
`main.ts`	The main entry point for your application. Compiles the application with the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser. You can also use the AOT compiler without changing any code by appending the `--aot` flag to the CLI `build` and `serve` commands.

`polyfills.ts`	Provides polyfill scripts for browser support.
`styles.sass`	Lists CSS files that supply styles for a project. The extension reflects the style preprocessor you have configured for the project.
`test.ts`	The main entry point for your unit tests, with some Angular-specific configuration. You don't typically need to edit this file.

New Angular projects use strict mode by default. If this is not desired you can opt-out when creating the project. For more information, see [Strict mode](#).

{@a app-src}

Inside the `src/` folder, the `app/` folder contains your project's logic and data. Angular components, templates, and styles go here.

`src/app/` FILES	PURPOSE
`app/app.component.ts`	Defines the logic for the application's root component, named `AppComponent`. The view associated with this root component becomes the root of the [view hierarchy](#) as you add components and services to your application.
`app/app.component.html`	Defines the HTML template associated with the root `AppComponent`.
`app/app.component.css`	Defines the base CSS stylesheet for the root `AppComponent`.
`app/app.component.spec.ts`	Defines a unit test for the root `AppComponent`.
`app/app.module.ts`	Defines the root module, named `AppModule`, that tells Angular how to assemble the application. Initially declares only the `AppComponent`. As you add more components to the app, they must be declared here.

Application configuration files

The application-specific configuration files for the root application reside at the workspace root level. For a multi-project workspace, project-specific configuration files are in the project root, under `projects/project-name/`.

Project-specific [TypeScript](#) configuration files inherit from the workspace-wide `tsconfig.json`.

APPLICATION-SPECIFIC CONFIG FILES	PURPOSE
`.browserslistrc`	Configures sharing of target browsers and Node.js versions among various front-end tools. See [Browserslist on GitHub](#) for more information.
`karma.conf.js`	Application-specific [Karma](#) configuration.
`tsconfig.app.json`	Application-specific [TypeScript](#) configuration, including TypeScript and Angular template compiler options. See [TypeScript Configuration](#) and [Angular Compiler Options](#).
`tsconfig.spec.json`	[TypeScript](#) configuration for the application tests. See [TypeScript Configuration](#).

{@a multiple-projects}

Multiple projects

A multi-project workspace is suitable for an enterprise that uses a single repository and global configuration for all Angular projects (the "monorepo" model). A multi-project workspace also supports library development.

Setting up for a multi-project workspace

If you intend to have multiple projects in a workspace, you can skip the initial application generation when you create the workspace, and give the workspace a unique name. The following command creates a workspace with all of the workspace-wide configuration files, but no root-level application.

ng new my-workspace --create-application false

You can then generate applications and libraries with names that are unique within the workspace.

cd my-workspace ng generate application my-first-app

Multiple project file structure

The first explicitly generated application goes into the `projects/` folder along with all other projects in the workspace. Newly generated libraries are also added under `projects/`. When you create projects this way, the file structure of the workspace is entirely consistent with the structure of the workspace configuration file, `angular.json`.

my-workspace/ ... (workspace-wide config files) projects/ (generated applications and libraries) my-first-app/ --(an explicitly generated application) ... --(application-specific config) src/ --(source and support files for application) my-lib/ --(a generated library) ... --(library-specific config) src/ --source and support files for library)

Library project files

When you generate a library using the CLI (with a command such as `ng generate library my-lib`), the generated files go into the `projects/` folder of the workspace. For more information about creating your own libraries, see Creating Libraries.

Libraries unlike applications have their own `package.json` configuration file.

Under the `projects/` folder, the `my-lib` folder contains your library code.

LIBRARY SOURCE FILES	PURPOSE
`src/lib`	Contains your library project's logic and data. Like an application project, a library project can contain components, services, modules, directives, and pipes.
`src/test.ts`	The main entry point for your unit tests, with some library-specific configuration. You don't typically need to edit this file.
`src/public-api.ts`	Specifies all files that are exported from your library.
`karma.conf.js`	Library-specific Karma configuration.
`ng-package.json`	Configuration file used by ng-packagr for building your library.
`package.json`	Configures npm package dependencies that are required for this library.
`tsconfig.lib.json`	Library-specific TypeScript configuration, including TypeScript and Angular template compiler options. See TypeScript Configuration.

`tsconfig.lib.prod.json`	Library-specific TypeScript configuration that is used when building the library in production mode.
`tsconfig.spec.json`	TypeScript configuration for the library tests. See TypeScript Configuration.