# Fundamentals

## Components & Templates: Component styles

Angular applications are styled with standard CSS. That means you can apply everything you know about CSS stylesheets, selectors, rules, and media queries directly to Angular applications.

Additionally, Angular can bundle component styles with components, enabling a more modular design than regular stylesheets.

This page describes how to load and apply these component styles.

You can run the [live example](https://angular.io/generated/live-examples/component-styles/stackblitz.html) / [download example](https://angular.io/generated/zips/component-styles/component-styles.zip) in Stackblitz and download the code from there.

## Using component styles

For every Angular component you write, you may define not only an HTML template, but also the CSS styles that go with that template, specifying any selectors, rules, and media queries that you need.

One way to do this is to set the styles property in the component metadata. The styles property takes an array of strings that contain CSS code. Usually you give it one string, as in the following example:

src/app/hero-app.component.ts

content\_copy@[Component](https://angular.io/api/core/Component)({

selector: 'app-root',

template: `

<h1>Tour of Heroes</h1>

<app-hero-main [hero]="hero"></app-hero-main>

`,

styles: ['h1 { font-weight: normal; }']

})

export class HeroAppComponent {

/\* . . . \*/

}

## Style scope

The styles specified in @[Component](https://angular.io/api/core/Component) metadata apply only within the template of that component.

They are not inherited by any components nested within the template nor by any content projected into the component.

In this example, the h1 style applies only to the HeroAppComponent, not to the nested HeroMainComponent nor to <h1> tags anywhere else in the application.

This scoping restriction is a styling modularity feature.

* You can use the CSS class names and selectors that make the most sense in the context of each component.
* Class names and selectors are local to the component and don't collide with classes and selectors used elsewhere in the application.
* Changes to styles elsewhere in the application don't affect the component's styles.
* You can co-locate the CSS code of each component with the TypeScript and HTML code of the component, which leads to a neat and tidy project structure.
* You can change or remove component CSS code without searching through the whole application to find where else the code is used.

## Special selectors

Component styles have a few special selectors from the world of shadow DOM style scoping (described in the [CSS Scoping Module Level 1](https://www.w3.org/TR/css-scoping-1) page on the [W3C](https://www.w3.org/) site). The following sections describe these selectors.

### :host

Use the :host pseudo-class selector to target styles in the element that hosts the component (as opposed to targeting elements inside the component's template).

src/app/hero-details.component.css

content\_copy:host {

display: block;

border: 1px solid black;

}

The :host selector is the only way to target the host element. You can't reach the host element from inside the component with other selectors because it's not part of the component's own template. The host element is in a parent component's template.

Use the function form to apply host styles conditionally by including another selector inside parentheses after :host.

The next example targets the host element again, but only when it also has the active CSS class.

src/app/hero-details.component.css

content\_copy:host(.active) {

border-width: 3px;

}

### :host-context

Sometimes it's useful to apply styles based on some condition outside of a component's view. For example, a CSS theme class could be applied to the document <body> element, and you want to change how your component looks based on that.

Use the :host-context() pseudo-class selector, which works just like the function form of :host(). The :host-context() selector looks for a CSS class in any ancestor of the component host element, up to the document root. The :host-context() selector is useful when combined with another selector.

The following example applies a background-color style to all <h2> elements inside the component, only if some ancestor element has the CSS class theme-light.

src/app/hero-details.component.css

content\_copy:host-context(.theme-light) h2 {

background-color: #eef;

}

### (deprecated) /deep/, >>>, and ::ng-deep

Component styles normally apply only to the HTML in the component's own template.

Applying the ::ng-deep pseudo-class to any CSS rule completely disables view-encapsulation for that rule. Any style with ::ng-deep applied becomes a global style. In order to scope the specified style to the current component and all its descendants, be sure to include the :host selector before ::ng-deep. If the ::ng-deep combinator is used without the :host pseudo-class selector, the style can bleed into other components.

The following example targets all <h3> elements, from the host element down through this component to all of its child elements in the DOM.

src/app/hero-details.component.css

content\_copy:host /deep/ h3 {

font-style: italic;

}

The /deep/ combinator also has the aliases >>>, and ::ng-deep.

Use /deep/, >>> and ::ng-deep only with emulated view encapsulation. Emulated is the default and most commonly used view encapsulation. For more information, see the [Controlling view encapsulation](https://angular.io/guide/component-styles#view-encapsulation) section.

The shadow-piercing descendant combinator is deprecated and [support is being removed from major browsers](https://www.chromestatus.com/features/6750456638341120) and tools. As such we plan to drop support in Angular (for all 3 of /deep/, >>> and ::ng-deep). Until then ::ng-deep should be preferred for a broader compatibility with the tools.

## Loading component styles

There are several ways to add styles to a component:

* By setting styles or styleUrls metadata.
* Inline in the template HTML.
* With CSS imports.

The scoping rules outlined earlier apply to each of these loading patterns.

### Styles in component metadata

You can add a styles array property to the @[Component](https://angular.io/api/core/Component) decorator.

Each string in the array defines some CSS for this component.

src/app/hero-app.component.ts (CSS inline)

content\_copy@[Component](https://angular.io/api/core/Component)({

selector: 'app-root',

template: `

<h1>Tour of Heroes</h1>

<app-hero-main [hero]="hero"></app-hero-main>

`,

styles: ['h1 { font-weight: normal; }']

})

export class HeroAppComponent {

/\* . . . \*/

}

Reminder: these styles apply only to this component. They are not inherited by any components nested within the template nor by any content projected into the component.

The Angular CLI command [ng generate component](https://angular.io/cli/generate) defines an empty styles array when you create the component with the --inline-style flag.

content\_copyng generate component hero-app --inline-style

### Style files in component metadata

You can load styles from external CSS files by adding a styleUrls property to a component's @[Component](https://angular.io/api/core/Component) decorator:

src/app/hero-app.component.ts (CSS in file)

src/app/hero-app.component.css

content\_copy@[Component](https://angular.io/api/core/Component)({

selector: 'app-root',

template: `

<h1>Tour of Heroes</h1>

<app-hero-main [hero]="hero"></app-hero-main>

`,

styleUrls: ['./hero-app.component.css']

})

export class HeroAppComponent {

/\* . . . \*/

}

Reminder: the styles in the style file apply only to this component. They are not inherited by any components nested within the template nor by any content projected into the component.

You can specify more than one styles file or even a combination of styles and styleUrls.

When you use the Angular CLI command [ng generate component](https://angular.io/cli/generate) without the --inline-style flag, it creates an empty styles file for you and references that file in the component's generated styleUrls.

content\_copyng generate component hero-app

### Template inline styles

You can embed CSS styles directly into the HTML template by putting them inside <[style](https://angular.io/api/animations/style)> tags.

src/app/hero-controls.component.ts

content\_copy@[Component](https://angular.io/api/core/Component)({

selector: 'app-hero-controls',

template: `

<[style](https://angular.io/api/animations/style)>

button {

background-color: white;

border: 1px solid #777;

}

</[style](https://angular.io/api/animations/style)>

<h3>Controls</h3>

<button (click)="activate()">Activate</button>

`

})

### Template link tags

You can also write <link> tags into the component's HTML template.

src/app/hero-team.component.ts

content\_copy@[Component](https://angular.io/api/core/Component)({

selector: 'app-hero-team',

template: `

<!-- We must use a relative URL so that the AOT compiler can find the stylesheet -->

<link rel="stylesheet" href="../assets/hero-team.component.css">

<h3>Team</h3>

<ul>

<li \*[ngFor](https://angular.io/api/common/NgForOf)="let member of hero.team">

{{member}}

</li>

</ul>`

})

When building with the CLI, be sure to include the linked style file among the assets to be copied to the server as described in the [CLI wiki](https://github.com/angular/angular-cli/wiki/stories-asset-configuration).

Once included, the CLI will include the stylesheet, whether the link tag's href URL is relative to the application root or the component file.

### CSS @imports

You can also import CSS files into the CSS files using the standard CSS @import rule. For details, see [@import](https://developer.mozilla.org/en/docs/Web/CSS/@import) on the [MDN](https://developer.mozilla.org/) site.

In this case, the URL is relative to the CSS file into which you're importing.

src/app/hero-details.component.css (excerpt)

content\_copy/\* The AOT compiler needs the `./` to show that this is local \*/

@import './hero-details-box.css';

### External and global style files

When building with the CLI, you must configure the angular.json to include all external assets, including external style files.

Register **global** style files in the styles section which, by default, is pre-configured with the global styles.css file.

See the [CLI wiki](https://github.com/angular/angular-cli/wiki/stories-global-styles) to learn more.

### Non-CSS style files

If you're building with the CLI, you can write style files in [sass](http://sass-lang.com/), [less](http://lesscss.org/), or [stylus](http://stylus-lang.com/) and specify those files in the @[Component.styleUrls](https://angular.io/api/core/Component#styleUrls) metadata with the appropriate extensions (.scss, .less, .styl) as in the following example:

content\_copy@[Component](https://angular.io/api/core/Component)({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.scss']

})

...

The CLI build process runs the pertinent CSS preprocessor.

When generating a component file with ng generate component, the CLI emits an empty CSS styles file (.css) by default. You can configure the CLI to default to your preferred CSS preprocessor as explained in the [CLI wiki](https://github.com/angular/angular-cli/wiki/stories-css-preprocessors).

Style strings added to the @[Component.styles](https://angular.io/api/core/Component#styles) array must be written in CSS because the CLI cannot apply a preprocessor to inline styles.

## View encapsulation

As discussed earlier, component CSS styles are encapsulated into the component's view and don't affect the rest of the application.

To control how this encapsulation happens on a per component basis, you can set the view encapsulation mode in the component metadata. Choose from the following modes:

* ShadowDom view encapsulation uses the browser's native shadow DOM implementation (see [Shadow DOM](https://developer.mozilla.org/en-US/docs/Web/Web_Components/Shadow_DOM) on the [MDN](https://developer.mozilla.org/) site) to attach a shadow DOM to the component's host element, and then puts the component view inside that shadow DOM. The component's styles are included within the shadow DOM.
* Native view encapsulation uses a now deprecated version of the browser's native shadow DOM implementation - [learn about the changes](https://hayato.io/2016/shadowdomv1/).
* Emulated view encapsulation (the default) emulates the behavior of shadow DOM by preprocessing (and renaming) the CSS code to effectively scope the CSS to the component's view. For details, see [Inspecting generated CSS](https://angular.io/guide/component-styles#inspect-generated-css) below.
* None means that Angular does no view encapsulation. Angular adds the CSS to the global styles. The scoping rules, isolations, and protections discussed earlier don't apply. This is essentially the same as pasting the component's styles into the HTML.

To set the components encapsulation mode, use the encapsulation property in the component metadata:

src/app/quest-summary.component.ts

content\_copy// warning: few browsers support shadow DOM encapsulation at this time

encapsulation: [ViewEncapsulation.Native](https://angular.io/api/core/ViewEncapsulation#Native)

ShadowDom view encapsulation only works on browsers that have native support for shadow DOM (see [Shadow DOM v1](https://caniuse.com/#feat=shadowdomv1) on the [Can I use](http://caniuse.com/) site). The support is still limited, which is why Emulated view encapsulation is the default mode and recommended in most cases.

## Inspecting generated CSS

When using emulated view encapsulation, Angular preprocesses all component styles so that they approximate the standard shadow CSS scoping rules.

In the DOM of a running Angular application with emulated view encapsulation enabled, each DOM element has some extra attributes attached to it:

content\_copy<hero-details \_nghost-pmm-5>

<h2 \_ngcontent-pmm-5>Mister Fantastic</h2>

<hero-team \_ngcontent-pmm-5 \_nghost-pmm-6>

<h3 \_ngcontent-pmm-6>Team</h3>

</hero-team>

</hero-detail>

There are two kinds of generated attributes:

* An element that would be a shadow DOM host in native encapsulation has a generated \_nghost attribute. This is typically the case for component host elements.
* An element within a component's view has a \_ngcontent attribute that identifies to which host's emulated shadow DOM this element belongs.

The exact values of these attributes aren't important. They are automatically generated and you never refer to them in application code. But they are targeted by the generated component styles, which are in the <head> section of the DOM:

content\_copy[\_nghost-pmm-5] {

display: block;

border: 1px solid black;

}

h3[\_ngcontent-pmm-6] {

background-color: white;

border: 1px solid #777;

}

These styles are post-processed so that each selector is augmented with \_nghost or \_ngcontent attribute selectors. These extra selectors enable the scoping rules described in this page.