

Attribute directives

Change the appearance or behavior of DOM elements and Angular components with attribute directives.

See the for a working example containing the code snippets in this guide.

Building an attribute directive

This section walks you through creating a highlight directive that sets the background color of the host element to yellow.

1. To create a directive, use the CLI command `ng generate directive`.

`ng generate directive highlight`

The CLI creates `src/app/highlight.directive.ts`, a corresponding test file `src/app/highlight.directive.spec.ts`, and declares the directive class in the `AppModule`.

The CLI generates the default `src/app/highlight.directive.ts` as follows:

The `@Directive()` decorator's configuration property specifies the directive's CSS attribute selector, `[appHighlight]`.

1. Import `ElementRef` from `@angular/core`. `ElementRef` grants direct access to the host DOM element through its `nativeElement` property.
2. Add `ElementRef` in the directive's `constructor()` to inject a reference to the host DOM element, the element to which you apply `appHighlight`.
3. Add logic to the `HighlightDirective` class that sets the background to yellow.

Directives *do not* support namespaces.

```
{@a apply-directive} ## Applying an attribute directive
```

1. To use the `HighlightDirective`, add a `<p>` element to the HTML template with the directive as an attribute.

Angular creates an instance of the `HighlightDirective` class and injects a reference to the `<p>` element into the directive's constructor, which sets the `<p>` element's background style to yellow.

```
{@a respond-to-user}
```

Handling user events

This section shows you how to detect when a user mouses into or out of the element and to respond by setting or clearing the highlight color.

1. Import `HostListener` from `'@angular/core'`.

1. Add two event handlers that respond when the mouse enters or leaves, each with the `@HostListener()` decorator.

Subscribe to events of the DOM element that hosts an attribute directive, the `<p>` in this case, with the `@HostListener()` decorator.

The handlers delegate to a helper method, `highlight()`, that sets the color on the host DOM element, `e1`.

The complete directive is as follows:

The background color appears when the pointer hovers over the paragraph element and disappears as the pointer moves out.

`{@a bindings} ## Passing values into an attribute directive`

This section walks you through setting the highlight color while applying the `HighlightDirective`.

1. In `highlight.directive.ts`, import `Input` from `@angular/core`.

1. Add an `appHighlight @Input()` property.

The `@Input()` decorator adds metadata to the class that makes the directive's `appHighlight` property available for binding.

1. In `app.component.ts`, add a `color` property to the `AppComponent`.
1. To simultaneously apply the directive and the color, use property binding with the `appHighlight` directive selector, setting it equal to `color`.

The `[appHighlight]` attribute binding performs two tasks:

- * applies the highlighting directive to the `<p>` element
- * sets the directive's highlight color with a property binding

Setting the value with user input

This section guides you through adding radio buttons to bind your color choice to the `appHighlight` directive.

1. Add markup to `app.component.html` for choosing a color as follows:
1. Revise the `AppComponent.color` so that it has no initial value.
1. In `highlight.directive.ts`, revise `onMouseEnter` method so that it first tries to highlight with `appHighlight` and falls back to `red` if `appHighlight` is `undefined`.
1. Serve your application to verify that the user can choose the color with the radio buttons.

```

{@a second-property}
```

Binding to a second property

This section guides you through configuring your application so the developer can set the default color.

1. Add a second `Input()` property to `HighlightDirective` called `defaultColor`.
1. Revise the directive's `onmouseenter` so that it first tries to highlight with the `appHighlight`, then with the `defaultColor`, and falls back to `red` if both properties are `undefined`.
1. To bind to the `AppComponent.color` and fall back to “violet” as the default color, add the following HTML. In this case, the `defaultColor` binding doesn't use square brackets, `[]`, because it is static.

As with components, you can add multiple directive property bindings to a host element.

The default color is red if there is no default color binding. When the user chooses a color the selected color becomes the active highlight color.

```
  
{@a ngNonBindable}
```

Deactivating Angular processing with NgNonBindable

To prevent expression evaluation in the browser, add `ngNonBindable` to the host element. `ngNonBindable` deactivates interpolation, directives, and binding in templates.

In the following example, the expression `{{ 1 + 1 }}` renders just as it does in your code editor, and does not display 2.

Applying `ngNonBindable` to an element stops binding for that element's child elements. However, `ngNonBindable` still lets directives work on the element where you apply `ngNonBindable`. In the following example, the `appHighlight` directive is still active but Angular does not evaluate the expression `{{ 1 + 1 }}`.

If you apply `ngNonBindable` to a parent element, Angular disables interpolation and binding of any sort, such as property binding or event binding, for the element's children.