Binding syntax: an overview

Contents

Data-binding is a mechanism for coordinating what users see, specifically with application data values. While you could push values to and pull values from HTML, the application is easier to write, read, and maintain if you turn these tasks over to a binding framework. You simply declare bindings between binding sources, target HTML elements, and let the framework do the rest.

> See the live example / download example for a working example containing the code snippets in this guide.

Angular provides many kinds of data-binding. Binding types can be grouped into three categories distinguished by the direction of data flow:

- From the *source-to-view*
- From *view-to-source*
- Two-way sequence: *view-to-source-to-view*

Type	Syntax	Category

Interpolation
Property
Attribute
Class
Style
	`{{expression}}`
	`[target]="expression"`
	`bind-target="expression"`
		One-way
		from data source
		to view target

Event	`(target)="statement"`
	`on-target="statement"`
		One-way
		from view target
		to data source

Two-way	`[(target)]="expression"`
	`bindon-target="expression"`
		Two-way

Binding types other than interpolation have a **target name** to the left of the equal sign, either surrounded by punctuation, `[]` or `()`, or preceded by a prefix: `bind-`, `on-`, `bindon-`.

The *target* of a binding is the property or event inside the binding punctuation: `[]`, `()` or `[()]`.

Every public member of a **source** directive is automatically available for binding. You don't have to do anything special to access a directive member in a template expression or statement.

Data-binding and HTML

In the normal course of HTML development, you create a visual structure with HTML elements, and you modify those elements by setting element attributes with string constants.

```
<div class="special">Plain old HTML</div>
<img src="images/item.png">
<button disabled>Save</button>
```

With data-binding, you can control things like the state of a button:

src/app/app.component.html

```
<!-- Bind button disabled state to `isUnchanged` property -->
<button [disabled]="isUnchanged">Save</button>
```

Notice that the binding is to the `disabled` property of the button's DOM element, **not** the attribute. This applies to data-binding in general. Data-binding works with *properties* of DOM elements, components, and directives, not HTML *attributes*.

HTML attribute vs. DOM property

The distinction between an HTML attribute and a DOM property is key to understanding how Angular binding works. **Attributes are defined by HTML. Properties are accessed from DOM (Document Object Model) nodes.**

- A few HTML attributes have 1:1 mapping to properties; for example, `id`.
- Some HTML attributes don't have corresponding properties; for example, `aria-*`.
- Some DOM properties don't have corresponding attributes; for example, `textContent`.

It is important to remember that *HTML attribute* and the *DOM property* are different things, even when they have the same name. In Angular, the only role of HTML attributes is to initialize element and directive state.

Template binding works with *properties* and *events*, not *attributes*.

When you write a data-binding, you're dealing exclusively with the *DOM properties* and *events* of the target object.

> This general rule can help you build a mental model of attributes and DOM properties: **Attributes initialize DOM properties and then they are done. Property values can change; attribute values can't.**
>
> There is one exception to this rule. Attributes can be changed by `setAttribute()`, which re-initializes corresponding DOM properties.

For more information, see the MDN Interfaces documentation which has API docs for all the standard DOM elements and their properties. Comparing the `<td>` attributes attributes to the `<td>` properties provides a helpful example for differentiation. In particular, you can navigate from the attributes page to the properties via "DOM interface" link, and navigate the inheritance hierarchy up to `HTMLTableCellElement`.

Example 1: an `<input>`

When the browser renders `<input type="text" value="Sarah">`, it creates a corresponding DOM node with a `value` property initialized to "Sarah".

```
<input type="text" value="Sarah">
```

When the user enters "Sally" into the `<input>`, the DOM element `value` *property* becomes "Sally". However, if you look at the HTML attribute `value` using `input.getAttribute('value')`, you can see that the *attribute* remains unchanged—it returns "Sarah".

The HTML attribute `value` specifies the *initial* value; the DOM `value` property is the *current* value.

To see attributes versus DOM properties in a functioning app, see the live example / download example especially for binding syntax.

Example 2: a disabled button

The `disabled` attribute is another example. A button's `disabled` *property* is `false` by default so the button is enabled.

When you add the `disabled` *attribute*, its presence alone initializes the button's `disabled` *property* to `true` so the button is disabled.

```
<button disabled>Test Button</button>
```

Adding and removing the `disabled` *attribute* disables and enables the button. However, the value of the *attribute* is irrelevant, which is why you cannot enable a button by writing `<button disabled="false">Still Disabled</button>`.

To control the state of the button, set the `disabled` *property*,

> Though you could technically set the `[attr.disabled]` attribute binding, the values are different in that the property binding requires to a boolean value, while its corresponding attribute binding relies on

whether the value is `null` or not. Consider the following:

```
<input [disabled]="condition ? true : false">
<input [attr.disabled]="condition ? 'disabled' : null">
```

Generally, use property binding over attribute binding as it is more intuitive (being a boolean value), has a shorter syntax, and is more performant.

To see the `disabled` button example in a functioning app, see the live example / download example especially for binding syntax. This example shows you how to toggle the disabled property from the component.

Binding types and targets

The **target of a data-binding** is something in the DOM. Depending on the binding type, the target can be a property (element, component, or directive), an event (element, component, or directive), or sometimes an attribute name. The following table summarizes the targets for the different binding types.

Type	Target	Examples

Property	Element property
	Component property
	Directive property
		`src`, `hero`, and `ngClass` in the following:
		`<img [src]="heroImageUrl">`
		`<app-hero-detail [hero]="currentHero"></app-hero-detail>`
		`<div [ngClass]="{'special': isSpecial}"></div>`

Event	Element event
	Component event
	Directive event
		`click`, `deleteRequest`, and `myClick` in the following:
		`<button (click)="onSave()">Save</button>`
		`<app-hero-detail (deleteRequest)="deleteHero()"></app-hero-detail>`
		`<div (myClick)="clicked=$event" clickable>click me</div>`

Two-way	Event and property	`<input [(ngModel)]="name">`
Attribute	Attribute	

	(the exception)	`<button [attr.aria-label]="help">help</button>`
Class	`class` property	`<div [class.special]="isSpecial">Special</div>`
Style	`style` property	`<button [style.color]="isSpecial ? 'red' : 'green'">`