Binding syntax

Data binding automatically keeps your page up-to-date based on your application's state. You use data binding to specify things such as the source of an image, the state of a button, or data for a particular user.

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

Data binding and HTML

Developers can customize HTML by specifying attributes with string values. In the following example, `class`, `src`, and `disabled` modify the `<div>`, `<img>`, and `<button>` elements respectively.

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

Use data binding to control things like the state of a button:

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

{@a html-attribute-vs-dom-property}

HTML attributes and DOM properties

Angular binding distinguishes between HTML attributes and DOM properties.

Attributes initialize DOM properties and you can configure them to modify an element's behavior. Properties are features of DOM 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`.

Remember that HTML attributes and DOM properties 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.

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

Example 1: an `<input>`

When the browser renders `<input type="text" value="Sarah">`, it creates a corresponding DOM node with a `value` property and initializes that `value` 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 especially for binding syntax.

Example 2: a disabled button

A button's `disabled` property is `false` by default so the button is enabled.

When you add the `disabled` attribute, you are initializing the button's `disabled` property to `true` which disables the button.

```
<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 instead.

Property and attribute comparison

Though you could technically set the `[attr.disabled]` attribute binding, the values are different in that the property binding must be 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">
```

The first line, which uses the `disabled` property, uses a boolean value. The second line, which uses the disabled attribute checks for `null` .

Generally, use property binding over attribute binding as a boolean value is easy to read, the syntax is shorter, and a property is more performant.

To see the `disabled` button example in a functioning application, see the . This example shows you how to toggle the disabled property from the component.

Types of data binding

Angular provides three categories of data binding according to the direction of data flow:

- From the source to view
- From view to source
- In a two way sequence of view to source to view

Type	Syntax	Category

Interpolation
Property
Attribute
Class
Style
	`<code-example>`
	`  {{expression}}`
	`  [target]="expression"`
	`</code-example>`	

```
</td>

<td>
  One-way<br>from data source<br>to
view target
</td>
<tr>
  <td>
    Event
  </td>
  <td>
    <code-example>
      (target)="statement"
    </code-example>
  </td>

  <td>
    One-way<br>from view target<br>to
data source
  </td>
</tr>
<tr>
  <td>
    Two-way
  </td>
  <td>
    <code-example>
      [(target)]="expression"
    </code-example>
  </td>
  <td>
    Two-way
  </td>
</tr>
```

Binding types other than interpolation have a target name to the left of the equal sign. The target of a binding is a property or event, which you surround with square brackets, `[]`, parentheses, `()`, or both, `[()]`.

The binding punctuation of `[]`, `()`, `[()]`, and the prefix specify the direction of data flow.

- Use `[]` to bind from source to view.
- Use `()` to bind from view to source.
- Use `[()]` to bind in a two way sequence of view to source to view.

Place the expression or statement to the right of the equal sign within double quotes, `""`. For more information see Interpolation and Template statements.

Binding types and targets

The target of a data binding can be a property, an event, or an attribute name. Every public member of a source directive is automatically available for binding in a template expression or statement. The following table summarizes

the targets for the different binding types.

Type	Target	Examples
Property	Element property Component property Directive property	`alt`, `src`, `hero`, and `ngClass` in the following:
Event	Element event Component event Directive event	`click`, `deleteRequest`, and `myClick` in the following:
Two-way	Event and property	
Attribute	Attribute (the exception)	
Class	`class` property	
Style	`style` property	