Two-way binding

Two-way binding gives components in your application a way to share data. Use two-way binding to listen for events and update values simultaneously between parent and child components.

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

Prerequisites

To get the most out of two-way binding, you should have a basic understanding of the following concepts:

- Property binding
- Event binding
- Inputs and Outputs

Two-way binding combines property binding with event binding:

BINDINGS	DETAILS
Property binding	Sets a specific element property.
Event binding	Listens for an element change event.

Adding two-way data binding

Angular's two-way binding syntax is a combination of square brackets and parentheses, `[()]`. The `[()]` syntax combines the brackets of property binding, `[]`, with the parentheses of event binding, `()`, as follows.

src/app/app.component.html

```
<app-sizer [(size)]="fontSizePx"></app-sizer>
```

How two-way binding works

For two-way data binding to work, the `@Output()` property must use the pattern, `inputChange`, where `input` is the name of the `@Input()` property. For example, if the `@Input()` property is `size`, the `@Output()` property must be `sizeChange`.

The following `sizerComponent` has a `size` value property and a `sizeChange` event. The `size` property is an `@Input()`, so data can flow into the `sizerComponent`. The `sizeChange` event is an `@Output()`, which lets data flow out of the `sizerComponent` to the parent

component.

Next, there are two methods, `dec()` to decrease the font size and `inc()` to increase the font size. These two methods use `resize()` to change the value of the `size` property within min/max value constraints, and to emit an event that conveys the new `size` value.

src/app/sizer.component.ts

```
export class SizerComponent {

  @Input()  size!: number | string;
  @Output() sizeChange = new EventEmitter<number>();

  dec() { this.resize(-1); }
  inc() { this.resize(+1); }

  resize(delta: number) {
    this.size = Math.min(40, Math.max(8, +this.size +
delta));
    this.sizeChange.emit(this.size);
  }
}
```

The `sizerComponent` template has two buttons that each bind the click event to the `inc()` and `dec()` methods. When the user clicks one of the buttons, the `sizerComponent` calls the corresponding method. Both methods, `inc()` and `dec()`, call the `resize()` method with a `+1` or `-1`, which in turn raises the `sizeChange` event with the new size value.

src/app/sizer.component.html

```
<div>
  <button type="button" (click)="dec()"
title="smaller">-</button>
  <button type="button" (click)="inc()"
title="bigger">+</button>
  <span [style.font-size.px]="size">FontSize:
{{size}}px</span>
</div>
```

In the `AppComponent` template, `fontSizePx` is two-way bound to the `SizerComponent`.

src/app/app.component.html

```
<app-sizer [(size)]="fontSizePx"></app-sizer>
<div [style.font-size.px]="fontSizePx">Resizable
Text</div>
```

In the `AppComponent`, `fontSizePx` establishes the initial `SizerComponent.size` value by setting the value to `16`.

src/app/app.component.ts

```
fontSizePx = 16;
```

Clicking the buttons updates the `AppComponent.fontSizePx`. The revised `AppComponent.fontSizePx` value updates the style binding, which makes the displayed text bigger or smaller.

The two-way binding syntax is shorthand for a combination of property binding and event binding. The `SizerComponent` binding as separate property binding and event binding is as follows.

src/app/app.component.html (expanded)

```
<app-sizer [size]="fontSizePx"
(sizeChange)="fontSizePx=$event"></app-sizer>
```

The `$event` variable contains the data of the `SizerComponent.sizeChange` event. Angular assigns the `$event` value to the `AppComponent.fontSizePx` when the user clicks the buttons.

TWO-WAY BINDING IN FORMS

Because no built-in HTML element follows the `x` value and `xChange` event pattern, two-way binding with form elements requires `NgModel`. For more information on how to use two-way binding in forms, see Angular NgModel.

Last reviewed on Mon Feb 28 2022