# Data Binding In Angular

Data-binding means communication between your typescript code of your component and your template which user sees. Suppose you have some business logic in your component typescript code to fetch some dynamic data from the server and want to display this dynamic data to the user via template because user sees only template, here we need some kind of binding between your typescript code and template (view).

Data-binding means communication between the TypeScript code of your component and your template which the user sees. Suppose, you have some business logic in your component TypeScript code to fetch some dynamic data from the server and want to display this dynamic data to the user via template because the user sees only the template. Here, we need some kind of binding between your TypeScript code and template (View). This is where data-binding comes into the picture in Angular because it is responsible for this communication.

Data-binding can be either one-way or two-way. Angular provides various types of data binding -

* String Interpolation
* Property Binding
* Event Binding
* Two-way binding

Let’s explore these bindings one by one.

**String Interpolation**

String Interpolation is a one-way data-binding which is used to output the data from a TypeScript code to HTML template (view). It uses the template expression in double curly braces to display the data from the component to the view.

For Example – {{ data }}

I have already created an Angular project using Angular CLI (which is attached to this article). We will use this same project for all future articles.

**app.component.ts**

1. **import** { Component } from '@angular/core';
3. @Component({
4. selector: 'app-root',
5. templateUrl: './app.component.html',
6. styleUrls: ['./app.component.css']
7. })
8. **export** **class** AppComponent {
9. title = 'Data binding using String Interpolation';
10. }

In the app.component.ts file, we have a variable called “title” which has a string value. We want to display this title value on the app component HTML view to the user.

**app.component.html**

1. <h2>
2. {{ title }}
3. </h2>

In the app.component.html file, we have used title variable for displaying title value. Now when this view will be rendered in the browser, title value will be displayed over the page.

**Output**

String Interpolation can resolve some other expressions. We have a few examples of it below –

**Example 1**

Let’s initialize two variables with two numbers and display their addition on the view.

1. **export** **class** AppComponent {
2. title = "Data binding using String Interpolation";
3. numberA: number = 10;
4. numberB: number = 20;
5. }

In the View, the expression will look like below –

 <p>Calculation is : {{ numberA + numberB }} </p>

**Output**

**Example 2**

Let's define a method in the TypeScript code and call it in the View.

1. **export** **class** AppComponent {
2. title = "Data binding using String Interpolation";
3. numberA: number = 10;
4. numberB: number = 20;
6. addTwoNumbers() {
7. **return** **this**.numberA + **this**.numberB;
8. }
9. }

Now, call this method in View and see the output.

<p>Addition is : {{ addTwoNumbers() }} </p>

**Output**

**Property Binding**

Property binding is also a one-way data binding, where we bind a property of a DOM element to a field which is a property we define in our component typescript code. Behind the scene, Angular converts string interpolation into property binding.

For Example – <img [src]=”imgUrl” />

However, we can use string interpolation here like <img src=”{{ imgUrl }}” /> , but property binding is always a lot cleaner and shorter syntax to bind image source.

My recommendation is when you want to simply display some dynamic data from a component on the view between headings like h1, h2, p etc, use string interpolation.

Suppose if you will use property binding for just displaying some value between h2 heading; the syntax will be –

<h2 [textContent]="title"></h2>

Here, we are binding the text content property of h2 DOM element with the title property that we defined in our component. You can see that this syntax obviously is longer than string interpolation one.

**Note**String Interpolation and Property binding both are one-way binding. That means if field value in the component change, Angular will automatically update the DOM. But any changes in the DOM will not be reflected back in the component.

**app.componnt.ts**

Let’s define a property named “imgUrl” and bind this with an image tag in the template.

1. **export** **class** AppComponent {
2. title = "Data binding using Property Binding";
3. imgUrl="https://avatars2.githubusercontent.com/u/20270535?s=40&v=4";
4. }

**app.component.html**

Use this property in the View and see the output.

1. <h2>{{ title }}</h2> <!-- String Interpolation -->
2. <img [src]="imgUrl" /> <!-- Property Binding -->

**Output**

**Event Binding**

Angular provides us with other types of binding, i.e., event binding, which is used to handle the events raised from the DOM like button click, mouse move etc. Let’s understand this with the help of an example –

Suppose we have a button in the HTML template and we want to handle the click event of this button. To implement event binding, we will bind click event of a button with a method of the component.

**app.component.html**

**app.component.ts**

1. **export** **class** AppComponent {
2. onSave(){
3. console.log("Save operation is performed!");
4. }
5. }

**Output**

When a user clicks on this button, the button click event will fire and onSave method will be called, which will log a message in the console.

**Access DOM event object using $event**

Sometimes we need to access the event object that is raised in the event handler. For example on mouse move, we need to access the x and y positions using DOM event object, then we have to pass $event as a parameter. Let’s understand this with the help of an example –

**app.component.html**

**app.component.ts**

1. **export** **class** AppComponent {
2. onSave($event){
3. console.log("Save button is clicked!", $event);
4. }
5. }

**Output**

**Event Bubbling**

All the DOM events bubble up to the DOM tree unless a handler prevents further bubbling. This is just standard event propagation mechanism in DOM. Let’s understand this with the help of an example.

Let’s proceed with the button example. I have a div wrapper around the button in component HTML and div has also a click event handler just to log some message if div is clicked.

**app.component.html**

1. <h2>Event Bubbling Demo</h2>
3. <!-- Event Bubbling -->
4. <div (click)="onDivClick()">
5. <button (click)="onSave($event)">Save</button>  <!-- Event Binding -->
6. </div>

**App.component.ts**

1. **export** **class** AppComponent {
2. onSave($event){
3. console.log("Save button is clicked!", $event);
4. }
5. onDivClick(){
6. console.log("DIV is clicked!");
7. }
8. }

**Output**

Now let’s see the output. When we click the button, there are two messages logged in the console. One is from the handler of the click event of the button and second is from the handler of the click event of the div.

This is what we call “Event Bubbling”, an event bubbles up the event of its parent element.

If we would have another div or another element as a wrapper around of the div element and we handle the click event of that element, our button event will bubble up against the events of entire DOM tree.

To stop this event bubbling, we use stopPropogation method.

1. **export** **class** AppComponent {
2. onSave($event){
3. $event.stopPropagation();       <!--Stop event bubbling -->
4. console.log("Save button is clicked!", $event);
5. }
6. onDivClick(){
7. console.log("DIV is clicked!");
8. }
9. }

And now, when we click on the Save button, only the onSave method will be executed and will see only a message coming from the handle of the click event of the button.

**Event Filtering**

Angular provides a feature called Event Filtering. Let’s understand this with the help of an example.

Suppose I have a textbox and I want to log the text (which is entered in the textbox) in the console on pressing enter. Below is a traditional way to implement it.

**app.component.html**

1. <h2>Event Filtering Demo</h2>
2. <input (keyup)="onPressEnter($event)" />

**app.component.ts**

1. **export** **class** AppComponent {
2. onPressEnter($event){
3. **if**($event.keyCode===13){
4. console.log("Entered text: ",$event.target.value);
5. }
6. }
7. }

**Output**

Angular provides a shorter and cleaner way to implement the same. Let’s implement it with Angular event filtering.

**app.component.html**

1. <h2>Event Filtering Demo</h2>
2. <input (keyup.enter)="onPressEnter($event)" />

Just had to add enter filter with keyup event.

**app.component.ts**

1. **export** **class** AppComponent {
2. onPressEnter($event){
3. console.log("Entered text: ",$event.target.value);
4. }
5. }

And now, you can see there is no need to write that key code (13) and compare with.

**Output**

**Two-Way Data Binding**

Angular provides a very nice feature; i.e., two-way data binding. As of now, we have seen how to bind component data to view using one-way bindings. That means any change in the template(view) will not be reflected in the component typescript code.

Now, two-way binding has a feature to update data from component to view and vice-versa.

Syntax - For two-way data binding, we combine property binding and event binding both. Also, we can call “Banana in the Box”.

[(ngModel)] = ”[property of your component]”

**Important Note**To implement two-way data binding, you need to enable the ngModel directive and this depends upon FormsModule in angular/forms package. So you need to add FormsModule in imports[] array in the AppModule.

**app.module.ts**

1. **import** { BrowserModule } from '@angular/platform-browser';
2. **import** { NgModule } from '@angular/core';
3. **import** {FormsModule} from '@angular/forms';
4. **import** { AppComponent } from './app.component';
6. @NgModule({
7. declarations: [
8. AppComponent
9. ],
10. imports: [
11. BrowserModule,
12. FormsModule
13. ],
14. providers: [],
15. bootstrap: [AppComponent]
16. })
17. **export** **class** AppModule { }

**app.component.html**

1. <h2>Two-way Binding Demo</h2>
2. <input [(ngModel)]="fullName" /> <br/><br/>
3. <p> {{fullName}} </p>

**app.component.ts**

1. **import** { Component } from "@angular/core";
3. @Component({
4. selector: "app-root",
5. templateUrl: "./app.component.html",
6. styleUrls: ["./app.component.css"]
7. })
8. **export** **class** AppComponent {
9. fullName: string = "Rahul";
10. }

Now, ngModel will trigger the input event and update the value of input textbox in our view and when we will change the value of the textbox, it will automatically update in the “fullName” property of our component and vice-versa (as shown with the help of including string interpolation inside a paragraph).