Angular Two Way Binding

**Angular Two Way Data Binding with Examples**

you will understand the following pointers in detail which are related to Angular Two Way data binding.

Two way data binding means that **changes made to our model in the component are propagated to the view and that any changes made in the view are immediately updated in the underlying component data**. The two way data binding is nothing but both property binding & event binding applied together.

Two-way data binding performs the following actions:

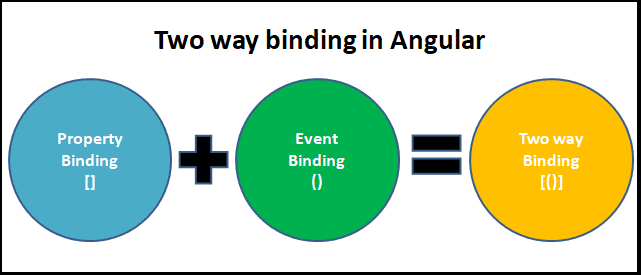
1. Sets a property of a component class
2. Listens for a DOM element change event

Angular v2+ supports two-way data binding using ngModel directive and also by having getter and setter methods.

## ngModel Directive

The ngModel directive with [()] syntax (also known as banana box syntax) syncs values from the UI to a property and vice-versa. So, whenever the user changes the value on UI, the corresponding property value will get automatically updated.

[()] = [] + () where [] binds attribute, and () binds an event.



**What is Angular Two Way Binding?**

The most popular and widely used data binding mechanism in Angular Application is two-way data binding. The two-way data binding is basically used in the input type filed or

any form element where the user type or provide any value or change any control value on the one side and on the other side, the same automatically updated into the component variables and vice-versa is also true.

The two-way data binding in Angular is actually a combination of [**Property Binding**](https://dotnettutorials.net/lesson/angular-property-binding/) and [**Event Binding**](https://dotnettutorials.net/lesson/angular-event-binding/). The Syntax is given below:

**<input [value] = ‘data 1’ (input) = ‘data = $event.target.value’>**

**Two-Way Binding using ngModel Directive:**

You can also implement the two-way data binding in Angular Application using the ngModel directive.

* The ngModel directive combines the square brackets of property binding with the parentheses of event binding in a single notation. T

he syntax to use ngModel for two-way data binding is given below.

* **<input [(ngModel)] = ‘data’>**

**Example to understand Angular Two Way Data Binding:**

Let’s directly start with an example. First, modify the **app.component.ts** file as shown below.

**import** **{** Component **}** from '@angular/core';

@Component**({**

selector: 'app-root',

template: `<div>

Name : <input [value]='Name'>

<br>

You entered : {{Name}}

</div>`

**})**

**export** **class** AppComponent **{**

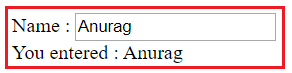
Name: **string** = 'Anurag';

**}**

From the above code, we need to understand two things.

1. **<input [value]=’Name’>**: it binds the component class “**Name**” property to the value property of the input element.
2. **You entered: {{Name}}:** The Interpolation displays the value we have in the “**Name**” property of the Component class on the web page.

So, when you browse the application, you should see the value Anurag in both the places as expected as shown in the below image.



At this moment if you change the value in the textbox, then that changed value is not going to be reflected in the browser (i.e. in the second line i.e. after you entered string). There are two different mechanisms that you can use to make them reflected (i.e. two-way data binding). Let us discuss one by one.

**Using the input event of the input control:**

One way to achieve two-way data binding is by binding to the input event of the input control. So, modify the **app.component.ts** file as shown below which and then we will understand the code.

**import** **{** Component **}** from '@angular/core';

@Component**({**

selector: 'app-root',

template: `<div>

Name : <input [value]='Name' (input) = 'Name = $event.target.value'>

<br>

You entered : {{Name}}

</div>`

**})**

**export** **class** AppComponent **{**

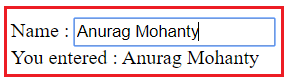
Name: **string** = 'Anurag';

**}**

Let’s understand the above code

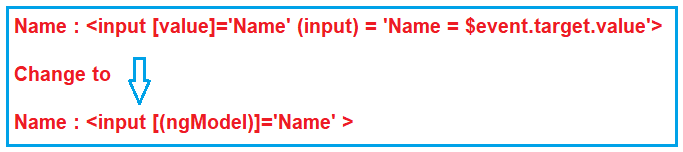
1. **[value]=’Name’**: This is property binding and it flows the data from the component class Name property to element property i.e. value property of the input element.
2. **(input)=’Name = $event.target.value’**: This is event binding and it flows the data in the opposite direction i.e. from the element to the component class property i.e. “Name”
3. **$event**: It is provided by angular event binding and it contains the event data. To retrieve the values from the input element, you need to use it as – **$event.target.value**.
4. **Name = $event.target.value**: This expression updates the value in the Name property of the component class
5. **You entered : {{name}}**: This interpolation expression will then display the updated value on the web page.

I hope now you understand the code present in the **app.component.ts** file. At this point, open the browser and start typing in the textbox. As you start typing that changed value will be reflected immediately on the page as shown in the below image.

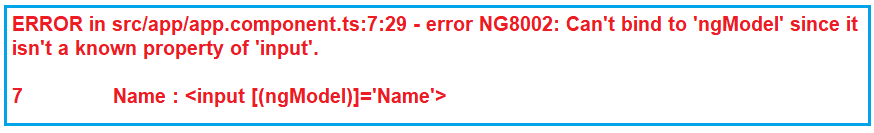


**Two Way Data binding using ngModel Directive:**

To simplify the two-way data binding, the angular framework has provided one directive called the ngModel directive. With the ngModel directive, you can change to existing code as shown below.



With the above changes in place in the app.component.ts file, at this point, if you run the application, then you will get the following error.



This is because the **ngModel**directive is available in the system module called **FormsModule**. If you want to use the **ngModel**directive, then in your root module that is **AppModule**, you will have to import the **FormsModule**first.

**Steps to use ngModel Directive:**

Here are the steps to import FormsModule into our AppModule

1. Open **app.module.ts** file  
2. Include the following import statement in it

**import { FormsModule } from ‘@angular/forms’;**  
3. Also, include FormsModule in the ‘imports’ array of @NgModule  
       **imports: [BrowserModule, FormsModule]**

With the above changes in place, the complete code of **App.Module.ts** is as follows.

**import** **{** NgModule **}** from '@angular/core';

**import** **{** AppRoutingModule **}** from './app-routing.module';

**import** **{** AppComponent **}** from './app.component';

**import** **{** FormsModule **}** from '@angular/forms';

@NgModule**({**

declarations: **[**

AppComponent

**]**,

imports: **[**

BrowserModule,

AppRoutingModule,

FormsModule

**]**,

providers: **[]**,

bootstrap: **[**AppComponent**]**

**})**

**export** **class** AppModule **{** **}**

**The complete codes of app.component.ts file as follows:**

**import** **{** Component **}** from '@angular/core';

@Component**({**

selector: 'app-root',

template: `<div>

Name : <input [(ngModel)]='Name'>

<br>

You entered : {{Name}}

</div>`

**})**

**export** **class** AppComponent **{**

Name: **string** = 'Anurag';

**}**

With the above changes, now reload the web page and you should see everything is working as expected.

Example

import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

  template: `<div >

<p>TwO  Way DataBinding</p>

Name : <input [(ngModel)]='title'>

<br>

You entered : {{title}}

  </div>`,

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'DataBinding';

  greet(){

    event?.stopPropagation();

    alert("innerelement");

  }

  greet1(){

    alert("OuterElement");

  }

}

The following is a full example of two-way data binding using get and set methods.

Example: Two-way Data Binding

 Copy

import { Component, OnInit } from '@angular/core';

@Component({

selector: 'app-greet',

template: `

User Name: <input type="text" [(ngModel)]="userName" ><br/>

{{userName}}`

})

export class GreetComponent implements OnInit {

constructor() { }

private \_userName: string = "bill gates";

get userName(): string {

return this.\_userName;

}

set userName(val: string) {

//do some extra work here

this.\_userName = val;

}

ngOnInit(): void {

}

}

### Angular Custom Two-Way Data Binding

In the post [Angular Two-Way Data Binding With Examples](https://www.netjstech.com/2020/04/angular-two-way-data-binding-with-example.html) we saw how two way binding can be done using **ngModel** and it is a combination of both property binding and event binding. In this post we’ll see how to do custom two way data binding in Angular.

### Angular custom two way data binding

You can use a custom property for two way data binding, for example-

<app-counter [(counter)]="counterValue"></app-counter>

Here **counter** is a property that has two way data binding. In the component you should have a counter property that can be assigned a value and a corresponding event named counterChange. Note that this convention has to be followed, *the event name has to be ‘property name + change’*.

### Angular custom two way data binding example

Here is a simple example of custom two way data binding. There is a child component that displays the counter value and has buttons to decrement or increment counter value.

**counter.component.html**

<div>

<label>Value </label> {{counter}}

<hr/>

<button class="btn btn-primary" (click)="dec()" title="smaller">-</button>

<button class="btn btn-primary" (click)="inc()" title="bigger">+</button>

</div>

**counter.component.ts**

import { Component, Input, Output, EventEmitter } from '@angular/core';

@Component({

selector: 'app-counter',

templateUrl: './counter.component.html'

})

export class CounterComponent{

@Input() counter : number;

@Output() counterChange = new EventEmitter<number>();

dec(){

--this.counter;

this.counterChange.emit(this.counter);

}

inc(){

++this.counter;

this.counterChange.emit(this.counter);

}

}

In the component, counter property is decorated with @Input decorator which means [property binding](https://www.netjstech.com/2020/04/angular-property-data-binding.html) where value of counter is expected from the parent component through this binding.

There is also an [event binding](https://www.netjstech.com/2020/04/angular-event-binding-with-examples.html) done using counterChange. Whenever decrement or increment button is clicked value of counter is changed and that changed value is emitted using counterChange event emitter.

**app.component.ts**

import { Component } from '@angular/core';

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css'],

})

export class AppComponent{

counterValue = 5;

}

**app.component.html**

<div class="container">

<div class="row">

<div class="col-md-10">

<app-counter [(counter)]="counterValue"></app-counter>

<label>Value of the counter is </label> {{counterValue}}

</div>

</div>

</div>

Here AppComponent.counterValue is two-way bound to the CounterComponent. counter property of CounterComponent gets its initial value from AppComponent.counterValue ( counterValue = 5 in AppComponent). Clicking the decrement and increment buttons updates the AppComponent.counterValue via the two-way binding.

So, there is a property binding where child property is bound to the parent property and there is also an event binding from child to parent causing the change in counter value.

**Initial screen**

On clicking decrement.

That's all for this topic **Angular Custom Two-Way Data Binding**.