Angular Event Binding

**Angular Event Binding with Examples**

To Practice the **Angular** **Event Binding**with example.

As of now, the bindings that we have discussed in this flow the data in one direction

i.e. from a component class property to an HTML element property.

But if you want to flow the data in the opposite direction

i.e. from HTML Element to Component, then you need to use Event Binding. At the end of this

you will understand the following pointers in detail.

1. **What is Event Binding in Angular?**
2. **How Does Event Binding work in Angular?**
3. **Angular Event Binding Example**

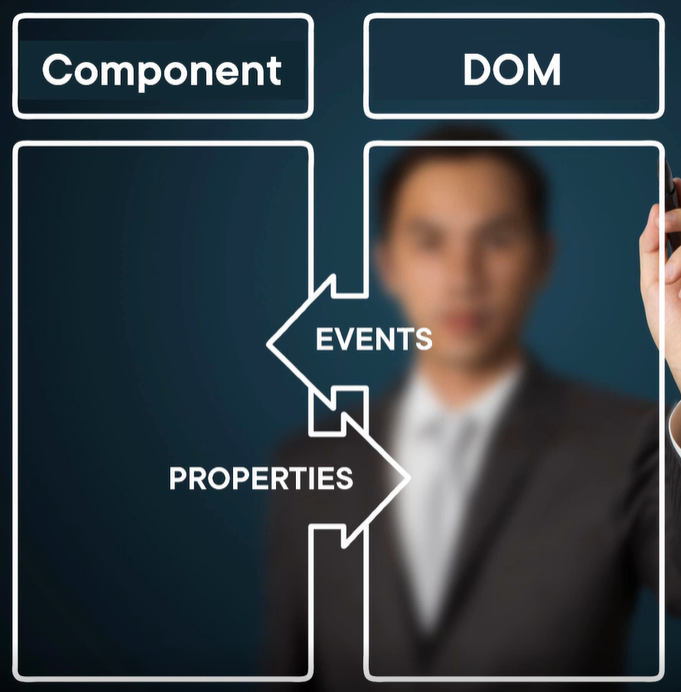
**What is Event Binding in Angular?**

When a user interacts with an application in the form of a keyboard movement, button click, mouse over,

selecting from a drop-down list,

typing in a textbox, etc.

it generates an event. These events need to be handled to perform some kind of action. This is where event binding comes into the picture and in Angular Application, we can use event binding to get notified when these events occur.



**How Does Event Binding work in Angular?**

Let us understand how Event Binding works in Angular Application with an example.

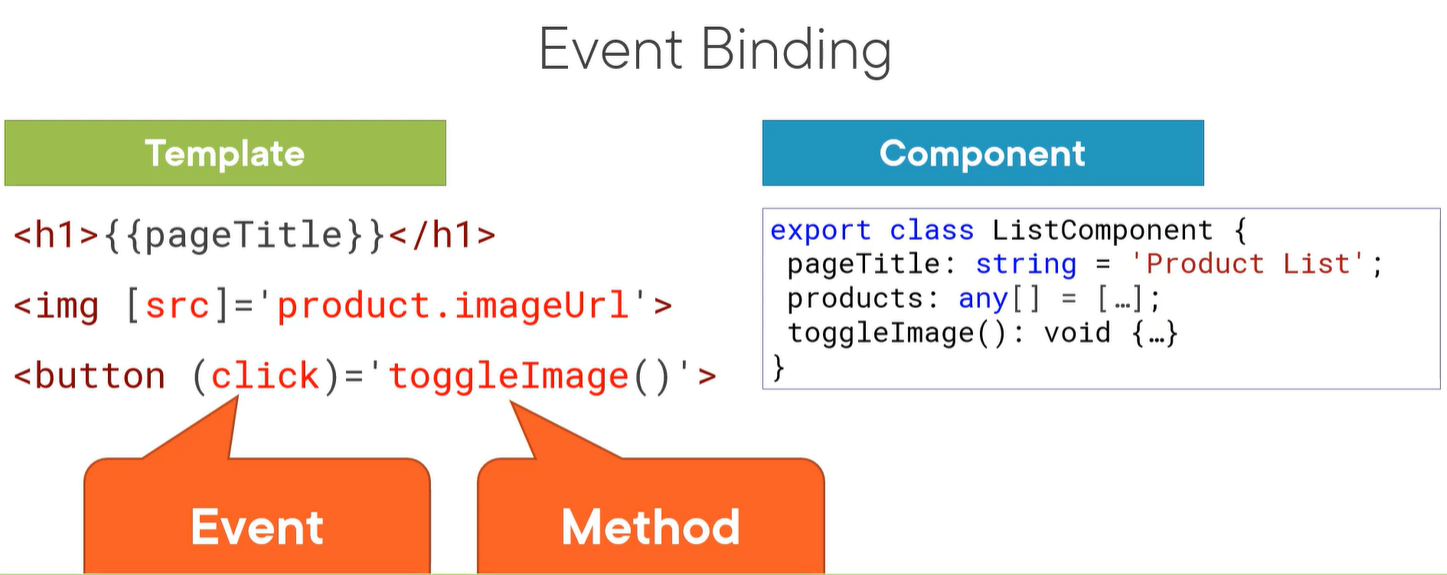
The following image shows the syntax for binding to the click event of a button. Within parentheses on the left-hand side of the equal sign, we have the target event, (click in this case) and on the right-hand side,

we have the template statement such as Component properties or methods. In this case, it is the component method i.e. **onClick**() method which is going to be called when the button click event occurs.

What is Event Binding in Angular?

With event binding, you can also use the on- prefix alternative as shown in the image below. This is known as the canonical form. It’s up to you which approach you follow. Behind the scene, they are going to perform the same task.

How Does Event Binding work in Angular?



# How event binding worksl

In an event binding, Angular configures an event handler for the target event. You can use event binding with your own custom events.

When the component or directive raises the event, the handler executes the template statement. The template statement performs an action in response to the event.

## Handling events

A common way to handle events is to pass the event object, $event, to the method handling the event. The $event object often contains information the method needs, such as a user's name or an image URL.

# The target event determines the shape of the $event object. If the target event is a native DOM element event, then $event is a [DOM event object](https://developer.mozilla.org/en-US/docs/Web/Events), with properties such as target and target.value. Event Binding in Angular 2 (v11)

Events are handled in Angular using the following special syntax.

(target event name) = "template statement"

Bind the target event name within parentheses on the left of an equal sign, and event handler method or statement on the right.

Example: Binding Button Click Event

 Copy

<button (click)="onShow()">Show</button>

Above, (click) binds the button click event and onShow() statement calls the onShow() method of a component.

Example: Handle Button Click Event in Component

 Copy

@Component({

selector: 'event-demo,

template: '<button (click)="onShow()" >Show</button>'

})

export class EventBindingDemoComponent implements OnInit {

constructor() { }

ngOnInit(): void {

}

onShow() {

alert('Show button clicked!');

}

}

Alternatively, use the on- prefix, known as the canonical form:

Example: on-event

<button on-click="onShow()" >Show</button>

By default, an event propagates up to the parent container event. In the following example, click event propagates to click of div and will call both the onShow() and onDivClick() methods.

Example: Event Bubbling

<div (click)="onDivClick()">

<button (click)="onShow()" >Show</button>

</div>

## $event

Mostly, when an event is raised, you may need to pass some value to the event handler function. This value can be number, string, or an object that contains information about an event.

You can pass the number or string value to the event handler function, as shown below.

Example: Passing Event Data

<button (click)="onShow(20)">Show</button>

Angular includes $event that contains the information about an event. The type of $event depends on the target event, e.g., if the target event is a native DOM element event, then it is an object.

Example: $event

<button (click)="onShow($event)">Show</button>

A component should define the onShow(event) method where the type of the parameter can be KeyboardEvent, MouseEvent, etc. If you don't know the exact event type, they use “any” type, as shown below.

Example: event Parameter

onShow(event:any) {

console.log(event);

}

If event is a native DOM element event then $event.target get DOM element reference using which you can access element's property e.g. $event.target.innerHTML returns the value of innerHTML property of a DOM element.

Example: Event Handling

<button (click)="onShow($event)">Show</button>

//component method

onShow(event:any) {

alert(event.target.innerHTML); // returns Show

}

You can use $event.target in the template statement. The following example binds a component property to $event.target.value of the input box on the input event without using ngModel.

Example: Bind Event without ngModel

<input type="text" (input)="userName=$event.target.value"><br/>

{{userName}}

In the following example the code sets the <input> value property by binding to the name property.

src/app/app.component.html

content\_copy<input [value]="currentItem.name"

(input)="currentItem.name=getValue($event)">

With this example, the following actions occur:

1. The code binds to the input event of the <input> element, which allows the code to listen for changes.
2. When the user makes changes, the component raises the input event.
3. The binding executes the statement within a context that includes the DOM event object, $event.
4. Angular retrieves the changed text by calling getValue($event.target) and updates the name property.

If the event belongs to a directive or component, $event has the shape that the directive or component produces.

The type of $event.target is only EventTarget in the template. In the getValue() method, the target is cast to an HTMLInputElement to allow type-safe access to its value property.

content\_copygetValue(event: [Event](https://angular.io/api/router/Event)): string {

return (event.target as HTMLInputElement).value;

}

**IN Angular13 Event Binding**

## ****Angular event binding****

Event binding in Angular will have the target event name within parentheses on the left of the equal sign and a quoted template statement on the right side. This is because property binding uses the square brackets, while event binding uses parentheses.

When the user interacts with an application in the form of a keyboard movement a mouse click, it generates the event. These events need to be handled to perform some action. This is where an event binding comes into the picture.

2.8M

**Next**

**Stay**

### ****Syntax:****

(event)

See the following example.

Write the following code inside the **app.component.html**file.

<h2> Angular 12 Event binding </h2>

<button (click) = "onBam()">Click Btn</button>

We have used the button element and added the click event to it. The **onBam()** event is native in HTML but, we are using angular here, so with the help of Angular, we can remove the **on**from the event and write event in the parentheses, which is click.

Angular has the pattern for native dom events where it is looking for the event name without the on, so any native dom event that is named on x, you would bind it by leaving off the on prefix. This is because angular expects the onBam() method to be available in execution.

When a user clicks on the button, it will call the **onBam()**function. Here, you can name the function, whatever you want. Then it will return whatever onBam() function will return. So all the flow will be transferred to the onBam() function.

Inside the **app.component.ts**file, we need to define the onBam() function. So let’s do it.

*// app.component.ts*

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

@Component({

selector: 'app-root',

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

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

})

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

onClick() {

console.log('Bamm!!!');

}

}

When the user clicks the button inside the browser console, we can see the**Bam!!!**

We can also use the prefix on-; this is known as the canonical form in event binding.

*<!-- app.component.html -->*

<h2> Angular 12 Event binding </h2>

<button on-click = "onBam()">Click Btn</button>

Please save the file, and it works the same as the previous one.

## ****$event handling and event handling statements****

In event binding, we bind the event handler for a target event. Whenever we perform some operations, the event will be raised.

The event handler then executes a template statement. The template handler will have a receiver, operating based on the event received and then responding. One such response would be storing the value from view to the array in the component.

If the event is the native DOM element event, then the **$event** is a DOM element object with different properties like target and target.value.

Okay, let’s understand with an example.

Create a file called **App.ts** inside an **app**folder and write the following code.

*// App.ts*

**export** **class** App {

name: string;

price: number;

}

Now, import the file inside the **app.component.ts**file.

*// app.component.ts*

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

**import** { App } **from** './App';

@Component({

selector: 'app-root',

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

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

})

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

app: App = {

name: 'FB',

price: 22

};

}

Also, we have defined the property values of the App class.

Now, write the following code inside the **app.component.html**file.

<div>

<h2> Angular 12 Event binding </h2>

<input [value]="app.name"

(input)="app.name=$event.target.value" />

</div>

In the above example we can see ‘**app.name**‘ bound to **$event.target.value**.

That’s it for this tutorial.

**Angular Event Binding Example:**

Let us understand Angular Event Binding with an example. Please modify the **app.component.ts** file as shown below.

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

@Component**({**

selector: 'app-root',

template: `<div>

<button (click)="onClick()">Click Me </button>

</div>`

**})**

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

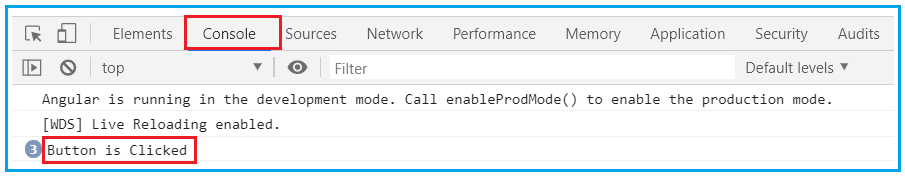
onClick**()**: **void** **{**

console.log**(**'Button is Clicked'**)**;

**}**

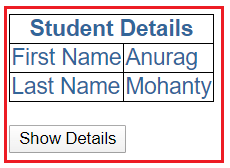
**}**

Now, run the application and launch the browser developer tools by pressing the F12 key. Once you open the browser developer tools click on the Console tab as shown in the image below. Notice every time you click the button, ‘Button is Clicked’ message is displayed on the console.

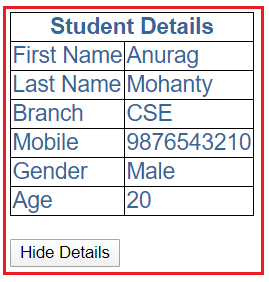


**Another Example:**

When the page loads for the first time, we want to display only the First Name and Last Name of the student. We also display the “Show Details” button as shown in the below image.



When the user clicks on the **“Show Details”** button, we want to display the “**Gender**“, “**Age**“, “**Mobile**”, and “**Branch**” as well. The text on the button should be changed to **“Hide Details”**as shown in the below image and when the user clicks on the **“Hide Details”** button, then the “**Gender**“, “**Age**“, “**Mobile**”, and “**Branch**”should be hidden and the button text should be changed to **“Show Details”**.



We can achieve this very easily in angular with the help of event binding. Here we will make use of one of the angular directives i.e. “**ngIf**“.

**Modify app.component.ts file:**

Notice we have introduced “ShowDetails” boolean property. The default value is false, so when the page loads for the first time, we will have “Gender”, “Age”, “Mobile”, and “Branch” hidden. We also have a method, ToggleDetails(), which will toggle the value of ShowDetails.

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

@Component**({**

selector: 'app-root',

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

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

**})**

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

ColumnSpan: **number** = 2;

FirstName: **string** = 'Anurag';

LastName: **string** = 'Mohanty';

Branch: **string** = 'CSE';

Mobile: **number** = 9876543210

Gender: **string** = 'Male';

Age: **number** = 20;

ShowDetails: **boolean** = **false**;

ToggleDetails**()**: **void** **{**

this.ShowDetails = !this.ShowDetails;

**}**

**}**

**Modify app.component.html file:**

Notice the click event of the button element is bounded to ToggleDetails() method. To dynamically change the text on the button, we are using a ternary operator:

**{{ShowDetails ? ‘Hide’ : ‘Show’}} Details**

We used ngIf structural directive on “Gender”, “Branch”, “Mobile” and “Age” <tr> elements. The \* prefix before a directive indicates, it is a structural directive. Besides ngIf, there are other structural directives which we will discuss in our upcoming articles.

The ngIf directive conditionally adds or removes content from the DOM based on whether or not an expression is true or false. If “**ShowDetails**” is true, “Gender”, “Branch”, “Mobile” and “Age” <tr> elements are added to the DOM, else removed.

**<**table**>**

**<**thead**>**

**<**tr**>**

**<**th attr.colspan="{{ColumnSpan}}"**>**

Student Details

**<**/th>

</tr**>**

**<**/thead>

<tbody>

<tr>

<td>First Name</td**>**

**<**td**>{{**FirstName**}}<**/td>

</tr**>**

**<**tr**>**

**<**td**>**Last Name**<**/td>

<td>{{LastName}}</td**>**

**<**/tr>

<tr \*ngIf='ShowDetails'>

<td>Branch</td**>**

**<**td**>{{**Branch**}}<**/td>

</tr**>**

**<**tr \*ngIf='ShowDetails'**>**

**<**td**>**Mobile**<**/td>

<td>{{Mobile}}</td**>**

**<**/tr>

<tr \*ngIf='ShowDetails'>

<td>Gender</td**>**

**<**td**>{{**Gender**}}<**/td>

</tr**>**

**<**tr \*ngIf='ShowDetails'**>**

**<**td**>**Age**<**/td>

<td>{{Age}}</td**>**

**<**/tr>

</tbody**>**

**<**/table>

<br/**>**

**<**button **(**click**)**='ToggleDetails()'**>**

**{{**ShowDetails ? 'Hide' : 'Show'**}}** Details

**<**/button**>**

**Modify app.component.css file:**

Modify the app.component.css file as shown below.

table **{**

color: *#369*;

font-family: Arial, Helvetica, sans-serif;

font-size:large;

border-collapse: collapse;

**}**

td **{**

border: 1px solid black;

**}**

thead**{**

border: 1px solid black;

**}**

Now run the application and you will see everything is working as expected as per our requirement.

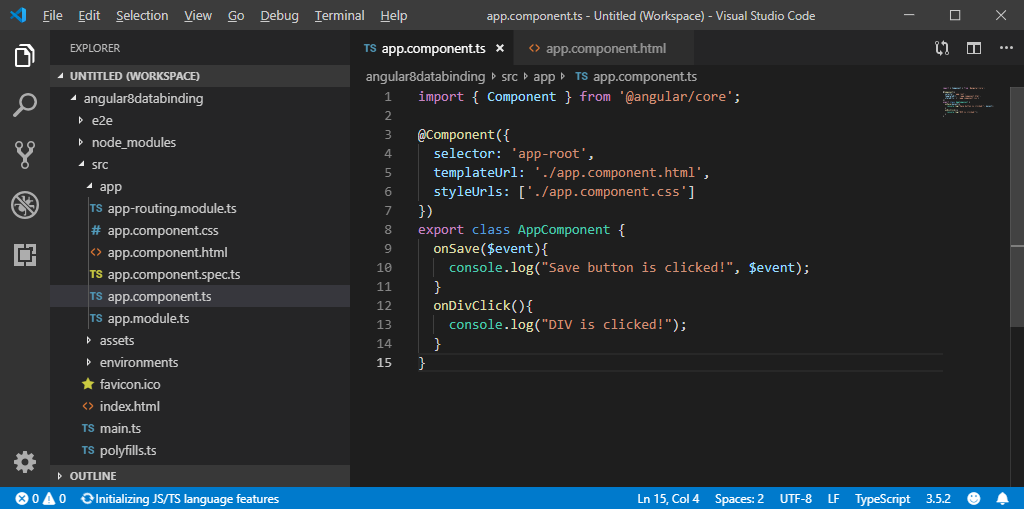
## Event Bubbling

Event bubbling is used to specify an order in which event handlers are called when one element is nested inside a second element, and both elements have registered a listener for the same event (i.e. click).

Let's see the above button example. Here, I have used a div wrapper around the button in component HTML and div has also a click event handler. It is only to show some message if div is clicked.

Use the following code in **app.component.ts** file:

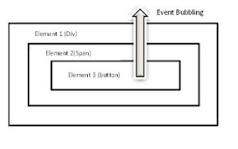
1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'app-root',
4. templateUrl: './app.component.html',
5. styleUrls: ['./app.component.css']
6. })
7. export class AppComponent {
8. onSave($event){
9. console.log("Save button is clicked!", $event);
10. }
11. onDivClick(){
12. console.log("DIV is clicked!");
13. }
14. }



**app.component.html:**

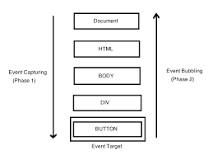
Event Bubbling. Event bubbling is **used to specify an order in which event handlers are called when one element is nested inside a second element, and both elements have registered a listener for the same event** (i.e. click).

What is meant by event bubbling?

[[](https://www.google.com/search?q=What+is+meant+by+event+bubbling?&tbm=isch&source=iu&ictx=1&vet=1&fir=UYoWvZpSwnIeFM%2Cn726l7iBIQa9rM%2C_&usg=AI4_-kTPVBnbItgGgjRhRZGpP_jKQc2-gg&sa=X&ved=2ahUKEwiGz4XVuN72AhURxDgGHVzUAjQQ9QF6BAgbEAE#imgrc=UYoWvZpSwnIeFM)](https://www.google.com/search?q=What+is+meant+by+event+bubbling?&tbm=isch&source=iu&ictx=1&vet=1&fir=UYoWvZpSwnIeFM%252Cn726l7iBIQa9rM%252C_&usg=AI4_-kTPVBnbItgGgjRhRZGpP_jKQc2-gg&sa=X&ved=2ahUKEwiGz4XVuN72AhURxDgGHVzUAjQQ9QF6BAgbEAE" \l "imgrc=UYoWvZpSwnIeFM)

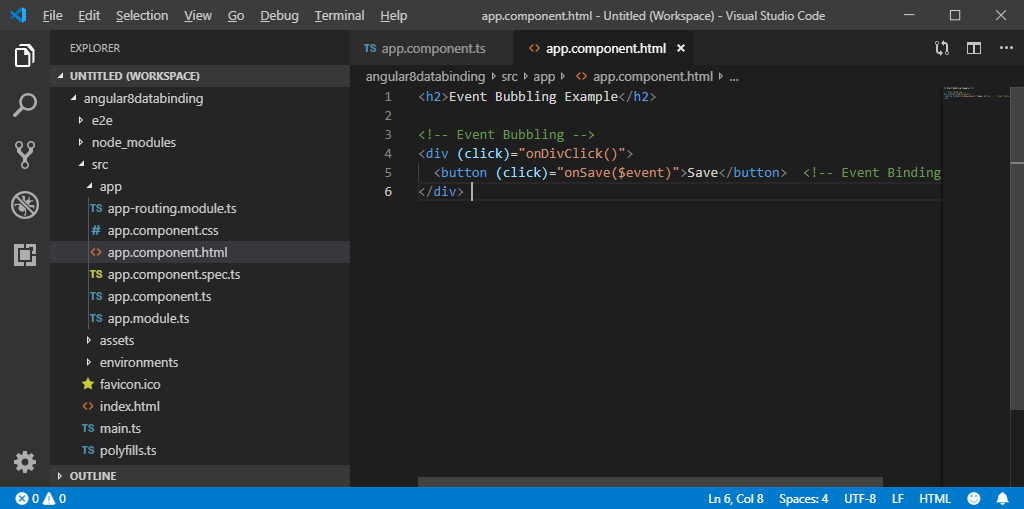
Event bubbling is a type of event propagation where the event first triggers on the innermost target element, and then successively triggers on the ancestors (parents) of the target element in the same nesting hierarchy till it reaches the outermost DOM element or document object (Provided the handler is initialized).

What is event bubbling and how do you stop it?

[[](https://www.google.com/search?q=What+is+event+bubbling+and+how+do+you+stop+it?&tbm=isch&source=iu&ictx=1&vet=1&fir=2xfEJGGf16aoCM%2Cpf6kmVdLFkpBZM%2C_&usg=AI4_-kSFnvLPUGUBsxirNtq_R3aaAUf7nw&sa=X&ved=2ahUKEwiGz4XVuN72AhURxDgGHVzUAjQQ9QF6BAgYEAE#imgrc=2xfEJGGf16aoCM)](https://www.google.com/search?q=What+is+event+bubbling+and+how+do+you+stop+it?&tbm=isch&source=iu&ictx=1&vet=1&fir=2xfEJGGf16aoCM%252Cpf6kmVdLFkpBZM%252C_&usg=AI4_-kSFnvLPUGUBsxirNtq_R3aaAUf7nw&sa=X&ved=2ahUKEwiGz4XVuN72AhURxDgGHVzUAjQQ9QF6BAgYEAE" \l "imgrc=2xfEJGGf16aoCM)

If you want to stop the event bubbling, **this can be achieved by the use of the event.** **stopPropagation() method**. If you want to stop the event flow from event target to top element in DOM, event. stopPropagation() method stops the event to travel to the bottom to top.

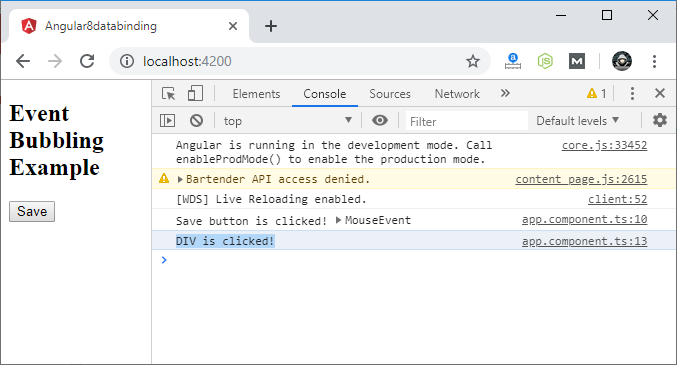
1. **<h2>**Event Bubbling Example**</h2>**
2. <!-- Event Bubbling -->
3. **<div** (click)="onDivClick()"**>**
4. **<button** (click)="onSave($event)"**>**Save**</button>**  <!-- Event Binding -->
5. **</div>**



**Output:**



Click on the "Save" button and open console to see result.



Here, you can see that your div message is also occurred. This is all due to event bubbling where you have specified onDivClick button.

**Example-EventBubbling**

@Component({

  selector: 'app-root',

  template: `<div (click)="greet1()">

  <button (click)="greet()">clickme</button>

  </div>`,

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'DataBinding';

  greet(){

    alert("innerelement");

  }

  greet1(){

    alert("OuterElement");

  }

}

**Example to stop EventBubbling**

@Component({

  selector: 'app-root',

  template: `<div (click)="greet1()">

  <button (click)="greet()">clickme</button>

  </div>`,

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'DataBinding';

  greet(){

    event?.stopPropagation();

    alert("innerelement");

  }

  greet1(){

    alert("OuterElement");

  }

}

Above example in Angular 11

import {Directive, HostListener} from "@angular/core";

@Directive({

selector: "[click-stop-propagation]"

})

export class ClickStopPropagation

{

@HostListener("click", ["$event"])

public onClick(event: any): void

{

event.stopPropagation();

}

}

# Stopping Angular Click Events from Propagating

We have a button inside the card:

<button (click)="issuesTab()">Issues</button>

Clicking this button also triggers onCardClick() . We want to prevent this.

## Approach

Pass $event :

<button (click)="issuesTab($event)">

And stop the propagation:

issuesTab(e:Event) {  
 e.stopPropagation()  
 window.open(ISSUES\_URL, '\_blank');  
}

Example to show or Hide Image using event Binding

imageUrl:string="https://media.istockphoto.com/photos/male-chimpanzee-in-business-clothes-picture-id184915585?k=20&m=184915585&s=612x612&w=0&h=iGwYUm5ZkmM14SBFzy1idVoVPVeNNrZ-KLmrtJ7k-EE=";

export class

isimageshown:boolean=false;

ImageToggle()

{

  this.isimageshown=!this.isimageshown;

}

Component.html

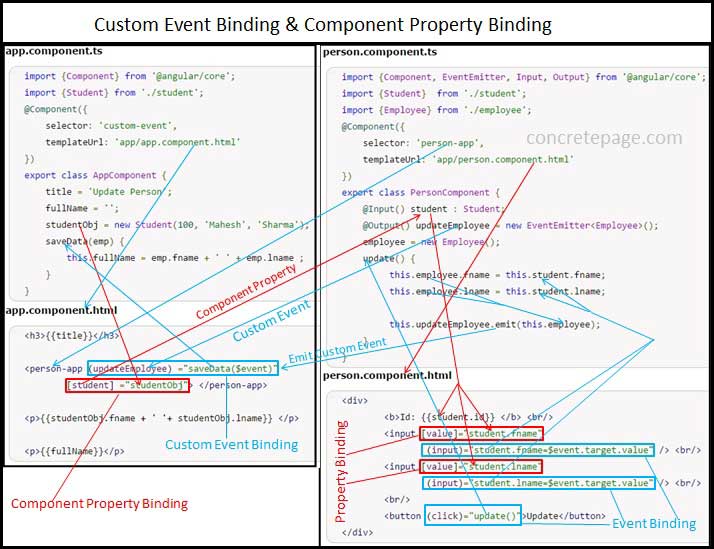
<img \*ngIf ="isimageshown" src={{imageUrl}} alt="programmer">

  </div>

<button (click)="ImageToggle()">showimage</button>

### Steps for Custom Event Binding & Component Property Binding with Diagram

Find the diagram for custom event binding and component property binding. Arrow in **light blue** color is showing flow of custom event binding and arrow in **red** color is showing flow of component property binding.



We will observe that the **light blue** arrow direction for custom event binding is from person.component.ts to app.component.ts and for component property binding **red** arrow direction is from app.component.ts to person.component.ts. Find the steps involved in custom event binding and component property binding.  
  
1. In the above diagram we are performing parent child communication using custom event binding and component property binding. app.component.ts is behaving as parent and person.component.ts is behaving as child. The child selector person-app will be used to create a custom element in parent component. Component property binding and custom event binding will take place in person-app element.  
  
2. In AppComponent we are creating an object of Student class as studentObj. In PersonComponent we are creating an input variable as student of Student class type. @Input decorator is used to mark a variable as input variable.  
  
3. In app.component.html we are performing component property binding between the objects studentObj of AppComponent and student of PersonComponent in person-app element.  
  
4. In person.component.html the object student is providing its value to text box using element property binding.  
  
5. When user changes values in text box then on input event the values of student object is getting changed. Here input event binding is taking place. $event.target.value returns current input value by user.  
  
6. In component property binding if the object sent to child is being updated then the reference object in parent will also get updated because they are pointing to the same object. So when student object of PersonComponent is getting updated then at the same time studentObj of AppComponent will also get updated. It means student and studentObj both objects are always in sync. In property binding it happens only in case of object of a class and not in case of primitive data type such string, number.  
  
7. On click of update button, update() method of PersonComponent will be called. Here click event binding is taking place.  
  
8. The object updateEmployee of angular EventEmitter class works as custom event binding name. @Output decorator is used to mark a variable as output variable. EventEmitter has emit() method that emits the payload.  
  
9. The execution of code updateEmployee.emit() will invoke custom event binding and will send employee object to custom event binding. Custom event is taking place in app.component.html . When updateEmployee event fires then it calls saveData() method of AppComponent. $event contains the payload sent by emit() method that is employee object.  
  
10. In the output we are printing studentObj and fullName of AppComponent using interpolation in app.component.html. Here studentObj and student object will keep same values even after student object changes by user update in text box.

### EventEmitter

EventEmitter is a class in angular framework. It has emit() method that emits custom events. We can use EventEmitter in custom event binding. To achieve it first we need to import it in our component file as given below.

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

And then initialize it using @Output decorator as follows.

@Output() updateEmployee = new EventEmitter<Employee>();

Here Employee is our TypeScript class. @Output() defines an output variable. updateEmployee will be a **custom event** name. Using emit() method of EventEmitter class we emits Employee object to parent component in custom event binding as follows.

this.updateEmployee.emit(this.employee);

The object emitted by emit() method can be accessed using event object $event.

### Create Custom Event

Suppose we want to create a custom event named as updateEmployee. So first of all we need to instantiate an instance of EventEmitter in the variable updateEmployee as below.

@Output() updateEmployee = new EventEmitter<Employee>();

We need to annotate it with @Output that declares it as output variable. Custom event binding is used in parent child component communication. Here we will send data from child component to parent component. To send data we need to use emit() method as below.

this.updateEmployee.emit(this.employee);

Now we are ready to use variable updateEmployee as custom event. Event binding can be done using parenthesis **()** or **on-** keyword. Find event binding using **()**.

<person-app (updateEmployee) ="saveData($event)"

[student] ="studentObj"> </person-app>

Now we will use **on-** keyword for updateEmployee event binding.

<person-app on-updateEmployee ="saveData($event)"

[student] ="studentObj"> </person-app>

In the above code snippet [student] ="studentObj" is working as a property binding to send data from parent component to child component.

### Create a Sample Class

For our demo we are creating two sample classes using TypeScript as follows.  
**student.ts**

export class Student {

constructor(public id: number, public fname: string, public lname: string) {

}

}

**employee.ts**

export class Employee {

public fname: string;

public lname: string;

constructor() {

}

}

We will create the instance of Student class in our parent component and will send it to child component using component property binding. Then we will set data to Employee class instance in child component and will send it to parent component using custom event binding.

### Create Child Component with EventEmitter

Find child component file used in our example.  
**person.component.ts**

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

import {Student} from './student';

import {Employee} from './employee';

@Component({

selector: 'person-app',

templateUrl: './person.component.html'

})

export class PersonComponent {

@Input() student : Student;

@Output() updateEmployee = new EventEmitter<Employee>();

employee = new Employee();

update() {

this.employee.fname = this.student.fname;

this.employee.lname = this.student.lname;

this.updateEmployee.emit(this.employee);

}

}

**@Input()**: Defines input variable in component communication. It is used to communicate from parent to child component using property binding.  
**@Output()**: Defines output variable in component communication. It is used to communicate from child to parent component using custom event binding.  
  
In our component file we have created a method update() that will call emit() method. update() method is being used in our HTML template as given below.  
**person.component.html**

<div>

<b>Id: {{student.id}} </b> <br/>

<input [value]="student.fname"

(input)="student.fname=$event.target.value" /> <br/>

<input [value]="student.lname"

(input)="student.lname=$event.target.value" /> <br/>

<br/>

<button (click)="update()">Update</button>

</div>

In our HTML template we are binding student object with HTML elements using property binding. We are also using DOM input event binding to fetch current value entered by user and assigning it back to student instance. To access the current values entered by user we use $event.target.value. On the click of button, update() method will be called that is using click event binding.

### Create Parent Component with Custom Event Binding

Now we will create parent component that will use custom event binding. Find the component.  
**app.component.ts**

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

import {Student} from './student';

@Component({

selector: 'app-root',

templateUrl: './app.component.html'

})

export class AppComponent {

title = 'Update Person';

fullName = '';

studentObj = new Student(100, 'Mahesh', 'Sharma');

saveData(emp) {

this.fullName = emp.fname + ' ' + emp.lname ;

}

}

Here we are creating an object of Student class. Method saveData() will be called in custom event binding. Find the HTML template used in our parent component.  
**app.component.html**

<h3>{{title}}</h3>

<person-app (updateEmployee) ="saveData($event)"

[student] ="studentObj"> </person-app>

<p>{{studentObj.fname + ' '+ studentObj.lname}} </p>

<p>{{fullName}}</p>

Look at the code, the variable updateEmployee declared in child component using EventEmitter will work as an event that is **custom event**. The event will be executed when the below line of code will execute.

this.updateEmployee.emit(this.employee);

The property student in our HTML template binds the value to input variable in child component.

### Create Module

**app.module.ts**

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

import {BrowserModule} from '@angular/platform-browser';

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

import {PersonComponent} from './person.component';

@NgModule({

imports: [BrowserModule],

declarations: [AppComponent, PersonComponent],

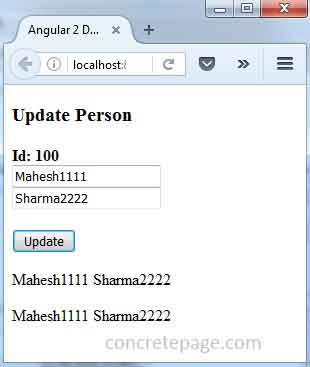
bootstrap: [AppComponent]

})

export class AppModule { }

### Run Application

To run the application, find the steps.  
**1.** Download source code using download link given below on this page.  
**2.** Use downloaded **src** in your Angular CLI application. To install Angular CLI, find the [link](https://angular.io/start).  
**3.** Run **ng serve** using command prompt.  
**4.** Access the URL **http://localhost:4200**  
Find the print screen of the output.



Here we are getting two output, first is because of component property binding and second is because of custom event binding.  
When we change first name and last name in text box then by input event the student object gets changed and because of component property binding it is the same object reference of studentObj that will also get changed.  
On click of **update** button update method of child component is called and hence emit() method will execute that will generate updateEmployee event in parent component and that will call saveData() method because of custom event bindin