Event Binding in Angular

## Event Binding

Event binding allows us to bind events such as keystroke, clicks, hover, touche, etc to a method in component. It is one way from view to component. By tracking the user events in the view and responding to it, we can keep our component in sync with the view. For Example, when the user changes to an input in a text box, we can update the model in the component, run some validations, etc. When the user submits the button, we can then save the model to the backend server.

## Syntax

The Angular event binding consists of two parts

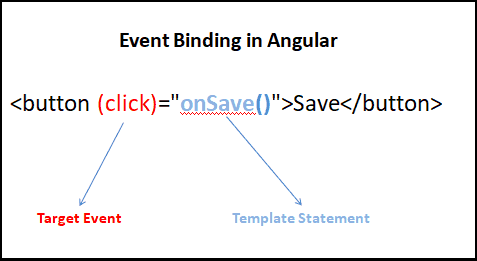
|  |  |
| --- | --- |
| 1  2  3 | (target-event)="TemplateStatement" |

* We enclose the target event name in parentheses on the left side
* Assign it to a template statement within a quote on the right side

Angular event binding syntax consists of a target event name within parentheses on the left of an equal sign, and a quoted template statement on the right.

The following event binding listens for the button’s click events, calling the component’s onSave() method whenever a click occurs

|  |  |
| --- | --- |
| 1  2  3 | <button (click)="onSave()">Save</button> |



## Event Binding Example

Create a new angular application

|  |  |
| --- | --- |
| 1  2  3 | ng new event |

Copy the following code to app.component.html

***app.component.html***

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | <h1 [innerText]="title"></h1>    <h2>Example 1</h2>  <button (click)="clickMe()">Click Me</button>  <p>You have clicked {{clickCount}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-1)

Add the following code to the app.component.ts

***app.component.ts***

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | clickCount=0    clickMe() {      this.clickCount++;    } |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-1)

In the above example, the component listens to the click event on the button. It then executes the clickMe() method and increases the clickCount by one.

## Template statements have side effects

Unlike the [Property Binding](https://www.tektutorialshub.com/angular/property-binding-in-angular/) & [Interpolation](https://www.tektutorialshub.com/angular/interpolation-in-angular/), where we use the template expression is used, in the case of event binding we use template statement.

The Template statement can change the state of the component. Angular runs the change detection and updates the view so as to keep it in sync with the component.

## on-

Instead of parentheses, you can also use the on- syntax as shown below.

|  |  |
| --- | --- |
| 1  2  3 | <button on-click="clickMe()">Click Me</button> |

## Multiple event handlers

You can also bind an unlimited number of event handlers on the same event by separating them with a semicolon :

Add a new component property

|  |  |
| --- | --- |
| 1  2  3 | clickCount1=0; |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-2?file=src/app/app.component.html)

And in the template, call clickMe() method and then an assignment clickCount1=clickCount

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | //Template    <h2>Example 2</h2>  <button (click)="clickMe() ; clickCount1=clickCount">Click Me</button>  <p>You have clicked {{clickCount}}</p>  <p>You have clicked {{clickCount1}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-2?file=src/app/app.component.html)

## $event Payload

DOM Events carries the event payload. I.e the information about the event. We can access the event payload by using $event as an argument to the handler function.

|  |  |
| --- | --- |
| 1  2  3  4 | <input (input)="handleInput($event)">  <p>You have entered {{value}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-3?file=src/app/app.component.ts)

And in the component

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | value=""  handleInput(event) {    this.value = (event.target as HTMLInputElement).value;  } |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-3?file=src/app/app.component.ts)

The properties of a $event object vary depending on the type of DOM event. For example, a mouse event includes different information than an input box editing event.

Remember you need to use the variable as $event in the Template statement. Example handleInput($event). Otherwise, it will result in an error

## Template reference variable

We can also make use of the template reference variable to pass the value instead of $event.

In the template

|  |  |
| --- | --- |
| 1  2  3  4  5 | <h2>Template Reference Variable</h2>  <input #el (input)="handleInput1(el)">  <p>You have entered {{val}}</p> |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-4?file=src/app/app.component.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | val="";  handleInput1(element) {    this.val=element.value;  } |

[***Source Code***](https://stackblitz.com/edit/event-binding-in-angular-ex-4?file=src/app/app.component.html)

## Key event filtering (with key.enter)

We use keyup/keydown events to listen for keystrokes. The following example does that

|  |  |
| --- | --- |
| 1  2  3  4 | <input (keyup)="value1= $any($event.target).value" />  <p>You entered {{value1}}</p> |

But Angular also offers a feature, where it helps to filter out certain keys. For Example, if you want to listen only to the enter keys you can do it easily

|  |  |
| --- | --- |
| 1  2  3  4 | <input (keyup.enter)="value2=$any($event.target).value">  <p>You entered {{value2}}</p> |

Here is an interesting example. On pressing enter key it updates the value3 variable and on escape clears the variable.

|  |  |
| --- | --- |
| 1  2  3  4 | <input (keyup.enter)="value3=$any($event.target).value" (keyup.escape)="$any($event.target).value='';value3=''">  <p>You entered {{value3}}</p> |

Note that we are using $any to cast $event.target to [any type](https://www.tektutorialshub.com/typescript/typescript-any-type/). Otherwise, the typescript will raise the error [Property ‘value’ does not exist on type ‘EventTarget’ Error in Angular](https://www.tektutorialshub.com/angular/property-value-does-not-exist-on-type-eventtarget-error-in-angular/)

Angular calls these pseudo-events.

You can also listen for the key combination

|  |  |
| --- | --- |
| 1  2  3  4 | <input (keyup.control.shift.enter)="value4=$any($event.target).value">  <p>You entered {{value4}}</p>    <h1 [innerText]="title"></h1>  <h2>Example 5</h2>  <input (keyup)="value1= $any($event.target).value" />  <p>You entered {{value1}}</p>  <input (keyup.enter)="value2=$any($event.target).value">  <p>You entered {{value2}}</p>  <input (keyup.enter)="value3=$any($event.target).value" (keyup.escape)="$any($event.target).value='';value3=''">  <p>You entered {{value3}}</p>  <input (keyup.control.shift.enter)="value4=$any($event.target).value">  <p>You entered {{value4}}</p>  <br />  <br />  import { Component, VERSION } from '@angular/core';  @Component({    selector: 'my-app',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {    title = 'Event Binding in Angular';    value1 = '';    value2 = '';    value3 = '';    value4 = '';  } |

## Custom events with EventEmitter

Directives & components can also raise events with [EventEmitter](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/). Using [EventEmiiiter](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) you can create a property and raise it using the EventEmitter.emit(payload). The Parent component can listen to these events using the event binding and also read the payload using the $event argument.

# Angular @input, @output & EventEmitter

**@input, @output & Eventemitter**

**@input**

Input decorator marks the property as the input property. I.e it can receive data from the parent component. The parent component uses the [property binding](https://www.tektutorialshub.com/angular/property-binding-in-angular/) to bind it to a component property. Whenever the value in the parent component changes angular updates the value in the child component.

**Example**

Consider the following component class

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @Component({    selector: 'app-customer-detail',    templateUrl: './customer-detail.component.html',    styleUrls: ['./customer-detail.component.css']  })  export class CustomerDetailComponent implements OnInit {    @Input() customer:Customer;  } |

We have Input decorator on the customer property. The component expects that the parent component will supply its value.

The parent component supplies the customer object using the [property binding](https://www.tektutorialshub.com/angular/property-binding-in-angular/) syntax. We add a square bracket around the customer property. Assign template expression (selectedCustomer) to it, which is a property in the parent component.

|  |  |
| --- | --- |
| 1  2  3 | <app-customer-detail [customer]="selectedCustomer"></app-customer-detail> |

**@output**

Output decorates the property as the output property. We initialize it as an EventEmitter. The child component raises the event and passes the data as the argument to the event. The parent component listens to events using [event binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) and reads the data.

**Example**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //Declare the property  @Output() customerChange:EventEmitter<Customer> =new EventEmitter<Customer>();    //Raise the event to send the data back to parent  update() {    this.customerChange.emit(this.customer);  } |

The customerChange is the Output property and is of type EventEmitter.

In the parent component, we subscribe to the event using the [event binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) syntax. Use the () around the event name (customerChange) and assign a template statement (update($event)) to it. It receives the data in the $event argument.

|  |  |
| --- | --- |
| 1  2  3 | <app-customer-detail [customer]="selectedCustomer" (customerChange)="update($event)"></app-customer-detail> |

Remember you must use the argument name as $event.

**EventEmitter**

EventEmitter is responsible for raising the event. The @output property normally is of type EventEmitter. The child component will use the emit() method to emit an event along with the data.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //Define  output property  @Output() customerChange:EventEmitter<Customer> =new EventEmitter<Customer>();    //Raise the event using the emit method.  update() {    this.customerChange.emit(this.customer);  } |

Now let us build an app to learn how to use Input, output & EventEmitter

[**BEST ANGULAR BOOKS**](https://www.tektutorialshub.com/angular/angular-best-books/)  
**The Top 8**[**Best Angular Books**](https://www.tektutorialshub.com/angular/angular-best-books/)**, which helps you to get started with Angular**

**@input, @output & Eventemitter Example**

The app we build has two components. The parent component shows a list of customers. The user has the option to click on the edit button, which results in a child component displaying the customer form Once the user updates the records, the child component raises the event. The parent captures the event. The parent then updates the list with the new data.

Create a new application using the following command

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ng new InputOutputExample      cd InputOutputExample |

Create the customerList & customerDetail components. Also, create the customer class

|  |  |
| --- | --- |
| 1  2  3  4  5 | ng g c customerList  ng g c customerDetail  ng g class customer |

**Customer**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | export class Customer {      customerNo: number=0;    name: string="";    address: string="";    city: string="";    state: string="";    country: string="";    } |

**app.module.ts**

The ngModel needs the FormsModule. Hence import it and add it in import metadata.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { FormsModule } from '@angular/forms'    import { AppRoutingModule } from './app-routing.module';  import { AppComponent } from './app.component';  import { CustomerListComponent } from './customer-list/customer-list.component';  import { CustomerDetailComponent } from './customer-detail/customer-detail.component';    @NgModule({    declarations: [      AppComponent,      CustomerListComponent,      CustomerDetailComponent    ],    imports: [      BrowserModule,      AppRoutingModule,      FormsModule    ],    providers: [],    bootstrap: [AppComponent],  })  export class AppModule { } |

**Child Component**

The child component gets an instance of the customer in its input property customer. The parent needs to set it using the [property binding](https://www.tektutorialshub.com/angular/property-binding-in-angular/)

Users can edit the customer. Once finished they will click the update button. The update method raises the customerChange event. We pass the customer as the argument to the event. The parent component listens to the event and receives the data.

The following is the complete code of the CustomerDetailComponent.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import { Component, OnInit, Input, Output,EventEmitter } from '@angular/core';  import { Customer } from '../customer';    @Component({    selector: 'app-customer-detail',    templateUrl: './customer-detail.component.html',    styleUrls: ['./customer-detail.component.css']  })  export class CustomerDetailComponent implements OnInit {      @Input() customer:Customer = new Customer();    @Output() customerChange:EventEmitter<Customer> =new EventEmitter<Customer>();      constructor() { }      ngOnInit() {    }      update() {      this.customerChange.emit(this.customer);    }    } |

'app-customer-detail' is the name of the selector for this component.

The customer property is the input property decorated with Input.

|  |  |
| --- | --- |
| 1  2  3 | @Input() customer:Customer = new Customer(); |

customerChange is decorated as the output property of type EventEmitter

|  |  |
| --- | --- |
| 1  2  3 | @Output() customerChange:EventEmitter<Customer> =new EventEmitter<Customer>(); |

Whenever the user updates the customer, we raise the event customerChange. We pass the updated customer as the argument to it.

|  |  |
| --- | --- |
| 1  2  3  4  5 | update() {    this.customerChange.emit(this.customer);  } |

The customer-detail.component.htmlis as follows.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | <p>Customer No : {{customer.customerNo}}</p>  <p>Name        : <input [(ngModel)]="customer.name"></p>  <p>Address     : <input [(ngModel)]="customer.address"></p>  <p>city     : <input [(ngModel)]="customer.city"></p>  <p>state     : <input [(ngModel)]="customer.state"></p>  <p>country     : <input [(ngModel)]="customer.country"></p>    <button (click)="update()">Update</button> |

The [ngModel](https://www.tektutorialshub.com/angular/ngmodel-two-way-data-binding-in-angular/) binds the customer to the input element. It is a [two-way binding](https://www.tektutorialshub.com/angular/ngmodel-two-way-data-binding-in-angular/). The click event of the button is bound to update() method in the component.

**Parent Component**

The job of the parent component is to display a list of customers. When the user clicks on the edit button pass the selected customer to the child component. Then wait for the customerChange event. Update the customer’s list on receipt of data from the child.

The following is the customer-list.component.html

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | <h2>List of Customers</h2>    <table class='table'>    <thead>      <tr>        <th>No</th>        <th>Name</th>        <th>Address</th>        <th>City</th>        <th>State</th>        <th>Country</th>        <th>Edit</th>      </tr>    </thead>    <tbody>      <tr \*ngFor="let customer of customers;">        <td>{{customer.customerNo}}</td>        <td>{{customer.name}}</td>        <td>{{customer.address}}</td>        <td>{{customer.city}}</td>        <td>{{customer.state}}</td>        <td>{{customer.country}}</td>        <td><button (click)="showDetails(customer)">Edit</button></td>      </tr>    </tbody>  </table>    <h3>Details</h3>  <app-customer-detail [customer]="selectedCustomer" (customerChange)="update($event)"></app-customer-detail> |

Use the [ngFor directive](https://www.tektutorialshub.com/angular/angular-ngfor-directive/) to loop through the customer list and display the customer details.

|  |  |
| --- | --- |
| 1  2  3 | <tr \*ngFor="let customer of customers;"> |

The event binding to capture the click event. We pass the customer object to the showDetails method

|  |  |
| --- | --- |
| 1  2  3 | <td><button (click)="showDetails(customer)">Edit</button></td> |

app-customer-detail is the selector for the CustomerDetailComponent. We use the [property binding](https://www.tektutorialshub.com/angular/property-binding-in-angular/) to send the selectedCustomer to the child component. The child component raises the customerChange event, which we listen to using the [event binding](https://www.tektutorialshub.com/angular/event-binding-in-angular/) and call the update method.

**Customer-list.component.ts**

The component code of the parent component. It has two method showDetails & update

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | import { Component, OnInit } from '@angular/core';  import { Customer } from '../customer';  import { element } from 'protractor';  import { ObjectUnsubscribedError } from 'rxjs';    @Component({    selector: 'app-customer-list',    templateUrl: './customer-list.component.html',    styleUrls: ['./customer-list.component.css']  })  export class CustomerListComponent implements OnInit {      customers: Customer[] = [        {customerNo: 1, name: 'Rahuld Dravid', address: '', city: 'Banglaore', state: 'Karnataka', country: 'India'},      {customerNo: 2, name: 'Sachin Tendulkar', address: '', city: 'Mumbai', state: 'Maharastra', country: 'India'},      {customerNo: 3, name: 'Saurrav Ganguly', address: '', city: 'Kolkata', state: 'West Bengal', country: 'India'},      {customerNo: 4, name: 'Mahendra Singh Dhoni', address: '', city: 'Ranchi', state: 'Bihar', country: 'India'},      {customerNo: 5, name: 'Virat Kohli', address: '', city: 'Delhi', state: 'Delhi', country: 'India'},      ]      selectedCustomer:Customer = new Customer();      constructor() { }      ngOnInit() {    }      showDetails(customer:Customer) {      this.selectedCustomer=Object.assign({},customer)    }      update(customer:Customer) {      console.log(customer)      var cust=this.customers.find(e => e.customerNo==customer.customerNo)      Object.assign(cust,customer)      alert("Customer Saved")    }  } |

The showDetails method gets the customer as its argument. We clone it & assign it to selectedCustomer

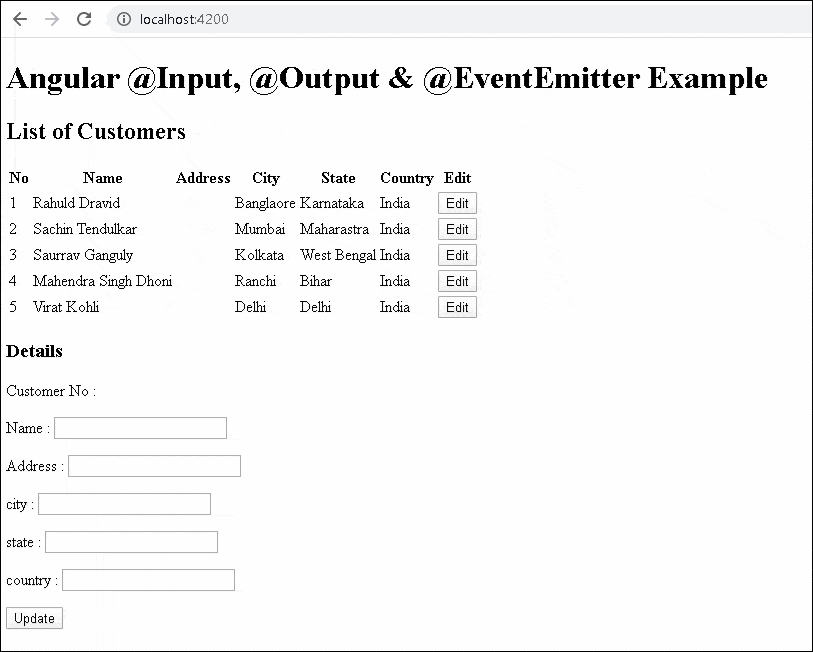
Since the customer is an object it is **Passed by Reference**. When you make any modification to the customer it will also be reflected in the customer’s collection. We want the update the customer’s only when we get the data from the child. Hence we clone the customer and send it to the child component.

If you are passing primitive data types like numbers are **Passed by Value**.

Finally in the root component (i.e. app.component.html ) copy the following

|  |  |
| --- | --- |
| 1  2  3 | <app-customer-list></app-customer-list> |

Run the app



**Notes on @Input & @Output**

**You can also pass the optional name**

Input decorator allows us to pass an option name, which you can use it while binding in the parent

For Example

|  |  |
| --- | --- |
| 1  2  3 | @Input(‘customerData’) customer:Customer; |

**Intercept input property changes with a setter**

You can also create a setter property

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | private \_customerData = '';    @Input()    set customer(customer: Customer) {      //You can add some custom logic here      this.\_customerData = customer;      console.log(this.\_customerData)    }    get customer(): string { return this.\_customerData; } |