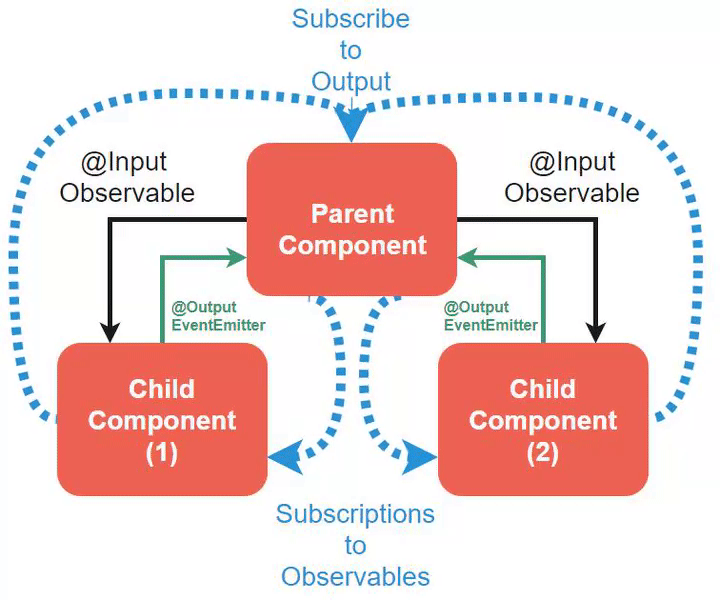
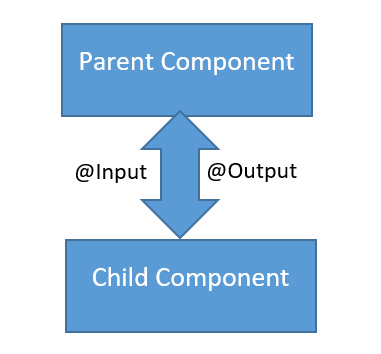
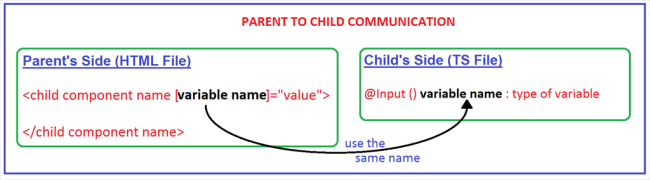
**3 ways to communicate data between Angular components**

* Parent to Child: Sharing Data via **Input**
* Child to Parent: Sharing Data via **ViewChild** with **AfterViewIni**t
* Child to Parent: Sharing Data via **Output()** and **EventEmitte**







## How to Pass data to a child component

In Angular, the Parent Component can communicate with the child component by setting its Property. To do that the Child component must expose its properties to the parent component. The Child Component does this by using the [**@Input**](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/)**decorator**

In the Child Component

1. Import the [@Input](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) module from @angular/Core Library
2. Mark those property, which you need data from the parent as input property using [@Input](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) decorator

let us start with passing data from the parent component to the child component. this can be done using the input property. @input decorator or input properties are used to pass data from parent to child component. to do this, we’ll need to modify child appchildcomponent as shown in the listing below:

## How to Pass data to a child component

In Angular, the Parent Component can communicate with the child component by setting its Property. To do that the Child component must expose its properties to the parent component. The Child Component does this by using the [**@Input**](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/)**decorator**

In the Child Component

1. Import the [@Input](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) module from @angular/Core Library
2. Mark those property, which you need data from the parent as input property using [@Input](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) decorator

In the Parent Component

1. **Bind the Child component property in the Parent Component when instantiating the Child**

Ad by Valueimpression

## @Input Decorator

**The @Input Decorator is used to configure the input properties of the component. This decorator as also supports change tracking.**

When you mark a property as input property, then the Angular injects values into the component property using [Property Binding](https://www.tektutorialshub.com/angular/angular-data-binding/#property-binding). The Property Binding uses the [] brackets.

The Binding Target (Property of the child component) is placed inside the square brackets. The Binding source is enclosed in quotes. [Property binding](https://www.tektutorialshub.com/angular/angular-data-binding/#property-binding) is one way from Component to the Target in the template

## @Input example

Now let us build a simple component to demonstrate the use of @Input.

Our application will have a counter which is incremented by the Parent Component. The Parent Component then passes this counter to the child component for display in its template

You can download the source of this tutorial from the [Github](https://github.com/tekTutorialsHub/Angular-Advanced-Components). The initial code is available in GettingStarted Folder and Final Code in InputDecorator folder

### The Child Component with @Input Decorator

Create the ChildComponent.ts under the src/app folder. And copy the following code

|  |  |
| --- | --- |
|  | import { Component, Input  } from '@angular/core';    @Component({      selector: 'child-component',      template: `<h2>Child Component</h2>                 current count is {{ count }}      `  })  export class ChildComponent {  **@Input()** count: number;  } |

Now, let us look at the code in detail

First, we import the @Input decorator from @angular/core

|  |  |
| --- | --- |
| 1  2  3 | import { Component, Input  } from '@angular/core'; |

We have defined the inline template for the child component, where it displays the current count.

|  |  |
| --- | --- |
|  | @Component({      selector: 'child-component',      template: `<h2>Child Component</h2>                 current count is {{ count }}     `  }) |

The Child Component expects the count to come from the Parent Component. Hence in ChildComponent decorate the count property with @Input decorator

|  |  |
| --- | --- |
| 1  2  3  4  5 | export class ChildComponent {      @Input() count: number;  } |

[**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**

### Bind to Child Property in Parent Component

Now, time to pass the Count values to the Child Component from the Parent

Open the app.component.ts and copy the following code

|  |  |
| --- | --- |
|  | import { Component} from '@angular/core';    @Component({    selector: 'app-root',    template: `          <h1>Welcome to {{title}}!</h1>          <button (click)="increment()">Increment</button>          <button (click)="decrement()">decrement</button>          <child-component [count]=Counter></child-component>` ,    styleUrls: ['./app.component.css']  })  export class AppComponent {    title = 'Component Interaction';    Counter = 5;      increment() {      this.Counter++;    }    decrement() {      this.Counter--;    }  } |

The inline template in the Parent Component has two buttons. The Buttons Increments/decrements the counter.

|  |  |
| --- | --- |
|  | <button (click)="increment()">Increment</button>      <button (click)="decrement()">decrement</button> |

In the next line, we are invoking the child component inside

|  |  |
| --- | --- |
|  | <child-component [count]=Counter></child-component> |

Here, we are using count property, which is a property of the child Component inside the square bracket. We bind it to Counter property of the Parent Component.

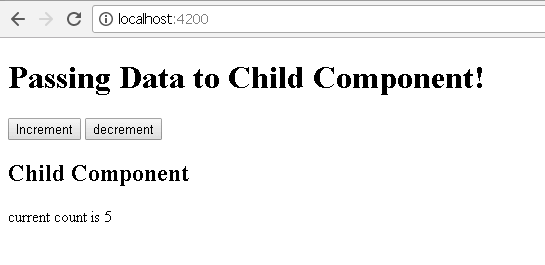
Remember square bracket represents the [Property Binding in Angular](https://www.tektutorialshub.com/angular/angular-data-binding/#property-binding).

Finally, we will add counter in Parent component and set it to 5 as shown below.

|  |  |
| --- | --- |
|  | export class AppComponent {    title = 'Component Interaction';    Counter = 5;      increment() {      this.Counter++;    }    decrement() {      this.Counter--;    }  } |

That’s it.

Now run the Code and you should see the following displayed in the browser



Click on Increment & Decrement buttons to see that the changes are propagated to the child component.

## Various ways to use @Input Decorator

We used input @Input Decorator to mark the property in child component as input property. There are two ways you can do it Angular.

1. Using the @Input decorator to decorate the class property
2. Using the input array meta data of the component decorator

### Using the @Input decorator to decorate the class property

We saw this in our above example.

|  |  |
| --- | --- |
|  | export class ChildComponent {       @Input() count: number;  } |

Using the input array metadata of the component decorator

The same result can be achieved by using Input array of the @Component decorator as shown below

|  |  |
| --- | --- |
|  | @Component({      selector: 'child-component',      inputs: ['count'],      template: `<h2>Child Component</h2>      current count is {{ count }}  `  })  export class ChildComponent {} |

We have moved the count property to inputs array of the component metadata.

### Aliasing input Property

You can Alias the input property and use the aliased name the parent component as shown below

|  |  |
| --- | --- |
|  | export class ChildComponent {       @Input('MyCount') count: number;  } |

Here, we are aliasing count property with MyCount alias

In the parent component, we can use the MyCount as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5 | template: `     <h1>Welcome to {{title}}!</h1>     <child-component [MyCount]=ClickCounter></child-component> |

## Detecting the Input changes

We looked at how to pass the data from parent to the child using @Input decorator and property binding.

Passing the data to child component is not sufficient, the child Component needs to know when the input changes so that it can act upon it.

There are two ways of detecting when input changes in the child component in Angular

1. Using [OnChanges](https://www.tektutorialshub.com/angular/angular-ngonchanges-life-cycle-hook/) LifeCycle Hook
2. Using Input Setter

Let us look at both the methods in detail

### Using OnChanges LifeCycle Hook

[ngOnChanges](https://www.tektutorialshub.com/angular/angular-ngonchanges-life-cycle-hook/) is a lifecycle hook, which angular fires when it detects changes to data-bound input property. This method receives a [SimpeChanges](https://www.tektutorialshub.com/angular/angular-ngonchanges-life-cycle-hook/" \l "simplechanges) object, which contains the current and previous property values. We can Intercept input property changes in the child component using this hook.

### How to use ngOnChanges for Change Detection

1. Import the OnChanges interface, SimpleChanges, SimpleChange from @angule/core library.
2. Implement the ngOnChanges() method. The method receives the SimpleChanges object containing the changes in each input property.

Let us update our Child Component to use the OnChanges hook

Open the child.component.ts and make the following changes

|  |  |
| --- | --- |
|  | import { Component, Input, OnChanges, SimpleChanges, SimpleChange  } from '@angular/core';    @Component({      selector: 'child-component',      template: `<h2>Child Component</h2>                 current count is {{ count }}      `  })  export class ChildComponent implements OnChanges {      @Input() count: number;        ngOnChanges(changes: SimpleChanges) {            for (let property in changes) {              if (property === 'count') {                console.log('Previous:', changes[property].previousValue);                console.log('Current:', changes[property].currentValue);                console.log('firstChange:', changes[property].firstChange);              }          }      }  } |

First, we are Importing the required libraries

|  |  |
| --- | --- |
|  | import { Component, Input, OnChanges, SimpleChanges, SimpleChange  } from '@angular/core'; |

Next, Modify the class to implement the Onchanges interface

|  |  |
| --- | --- |
|  | export class ChildComponent implements OnChanges { |

ngOnChanges method

|  |  |
| --- | --- |
|  | ngOnChanges(changes: SimpleChanges) {            for (let property in changes) {              if (property === 'count') {                console.log('Previous:', changes[property].previousValue);                console.log('Current:', changes[property].currentValue);                console.log('firstChange:', changes[property].firstChange);              }          }      } |

This method receives all the changes made to the input properties as SimpleChanges object. The SimpleChanges object whose keys are property names and values are instances of SimpleChange.

SimpleChange class Represents a basic change from a previous to a new value. It has three class members.

| **Property Name** | **Description** |
| --- | --- |
| previousValue:any | Previous value of the input property. |
| currentValue:any | New or current value of the input property. |
| FirstChange():boolean | Boolean value, which tells us whether it was the first time the change has taken place |

And we loop through the SimpleChanges to get our property count

Run the code and open the console log to watch the logs as you click on increment and decrement buttons in the parent component.

### Using Input Setter

We can use the property getter and setter to detect the changes made to the input property as shown below

In the Child Component create a private property called \_count

|  |  |
| --- | --- |
| 1  2  3 | private \_count = 0; |

Create getter & setter on property count and attach @Input Decorator. We intercept the input changes from the setter function and log them to console

|  |  |
| --- | --- |
|  | @Input()  set count(count: number) {      this.\_count = count;      console.log(count);  }  get count(): number { return this.\_count; } |

## Summary

In this tutorial, we looked at how to pass data from parent to child Component. The Child Component decorates the property using the @Input Decorator. In the Parent Component, we use property binding to bind it to the Property or method of Parent Component.

We can also track changes made to the Input Property either by Using hooking to ngOnChanges life cycle hook. Or using the Property setter

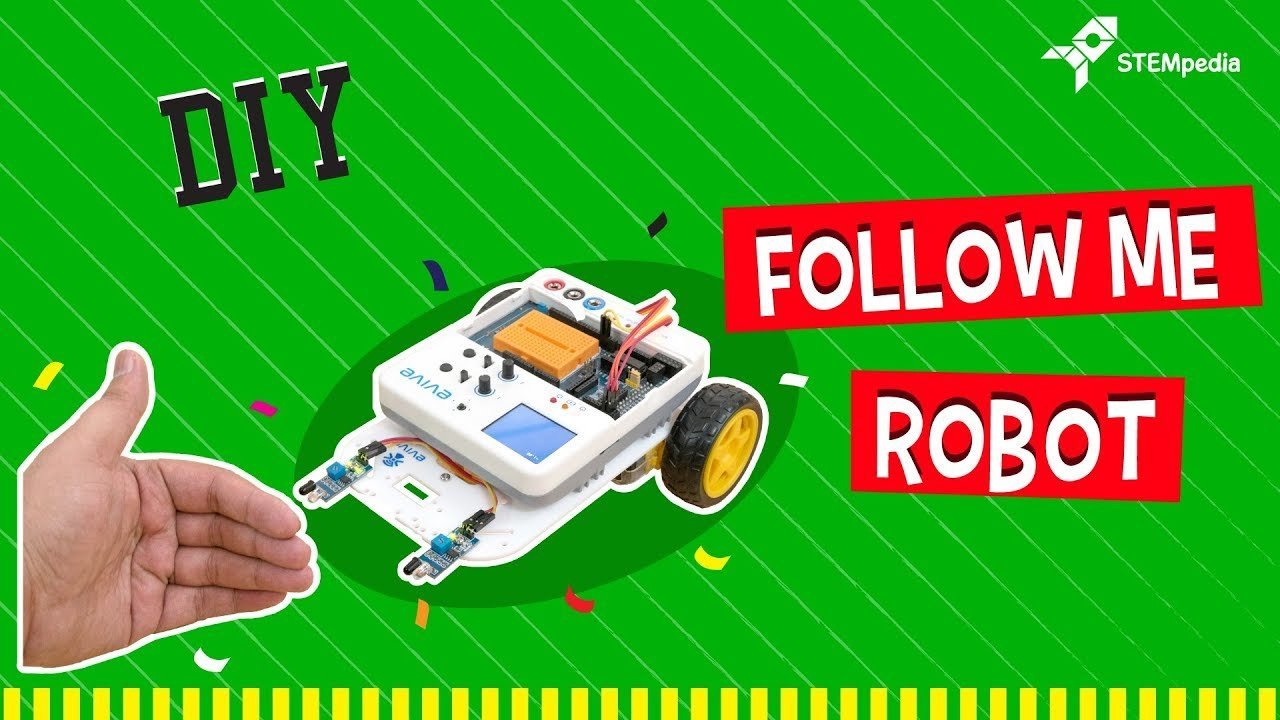
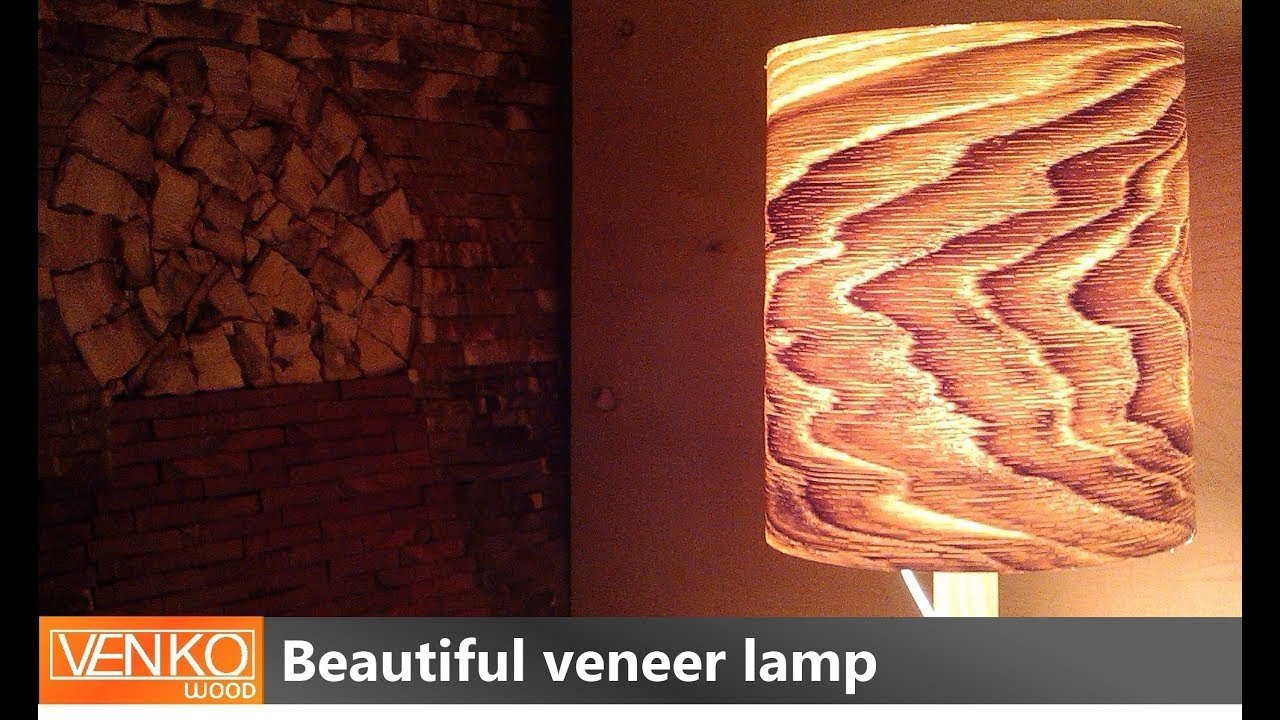
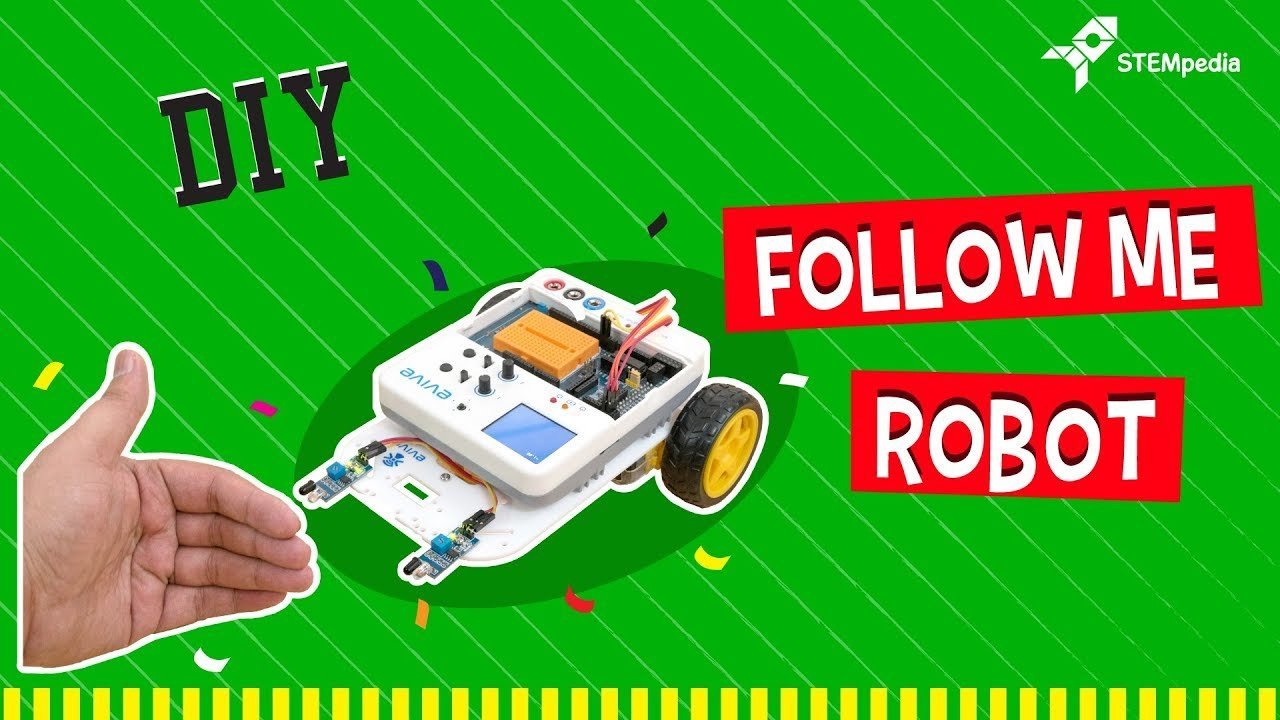
# **Angular Pass data from Child to parent component**

[19 Comments](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#comments) / 7 minutes of reading

[**Passing data to child Component**](https://www.tektutorialshub.com/angular/angular-passing-data-child-component/)

[**Life Cycle Hooks**](https://www.tektutorialshub.com/angular/angular-component-life-cycle-hooks/)

In this tutorial, we will learn how to Pass data from child to Parent Component in Angular.  In the previous tutorial, we looked at how the [pass data from parent to the child component](https://www.tektutorialshub.com/angular/angular-passing-data-child-component/) by setting its [input property](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/). The Child can send data to Parent by raising an event, Parent can interact with the child via local variable or Parent can call [@ViewChild](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/) on the child. We will look at all those options in this article.





00:18/05:12

Applies to: Angular 2 to the latest edition of i.e. Angular 8. Angular 9, Angular 10, Angular 11, Angular 12

**Table of Contents**

* [Pass data from Child to parent component](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#pass-data-from-child-to-parent-component)
* [Parent listens for child event](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#parent-listens-for-child-event)
  + [EventEmitter Property](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#eventemitter-property)
  + [@Output Decorator](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#output-decorator)
  + [How to Pass data to parent component using @Output](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#how-to-pass-data-to-parent-component-using-output)
  + [Passing data to parent component Via Events (Example)](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#passing-data-to-parent-component-via-events-example)
    - [Child Component](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#child-component)
    - [Parent Component](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#parent-component)
* [Parent uses local variable to access the Child in Template](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#parent-uses-local-variable-to-access-the-child-in-template)
  + [Child Component](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#child-component-2)
  + [Parent component](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#parent-component-2)
* [Parent uses a @ViewChild() to get reference to the Child Component](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#parent-uses-a-viewchild-to-get-reference-to-the-child-component)
  + [Child Component](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#child-component-3)
  + [Parent Component](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#parent-component-3)
* [Conclusion](https://www.tektutorialshub.com/angular/angular-pass-data-to-parent-component/#conclusion)

## Pass data from Child to parent component

**There are three ways in which the parent component can interact with the child component**

1. **Listens to Child Event**
2. **Uses**[**Local Variable**](https://www.tektutorialshub.com/angular/template-reference-variable-in-angular/)**to access the child**
3. **Uses a**[**@ViewChild**](https://www.tektutorialshub.com/angular/understanding-viewchild-viewchildren-querylist-in-angular/)**to get the reference to the child component**

Let us look at each of those scenarios in detail

## Parent listens for child event

The Child Component exposes an [EventEmitter](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) Property. This Property is adorned with the [@Output](https://www.tektutorialshub.com/angular/angular-input-output-eventemitter/) decorator. When Child Component needs to communicate with the parent it raises the event. The Parent Component listens to that event and reacts to it.

### EventEmitter Property

To Raise an event, the component must declare an EventEmmitter Property. The Event can be emitted by calling the .**emit() method**

For Example

|  |  |
| --- | --- |
|  | countChanged: EventEmitter<number> = new EventEmitter()//instantiate |

And then call emit method passing the whatever the data you want to send as shown below

|  |  |
| --- | --- |
| 1  2  3 | this.countChanged.emit(this.count); |

### @Output Decorator

Using the EventEmitter Property gives the components ability to raise an event. But to make that event accessible from parent component, you must decorate the property with @Output decorator

### How to Pass data to parent component using @Output

**In the child component**

1. Declare a property of type EventEmitter and instantiate it
2. Mark it with a @Output Decorator
3. Raise the event passing it with the desired data

**In the Parent Component**

1. Bind to the Child Component using [Event Binding](https://www.tektutorialshub.com/angular/angular-data-binding/#Event-Binding) and listen to the child events
2. Define the event handler function

### Passing data to parent component Via Events (Example)

Now let us build an application to demonstrate this

In the last [passing data to child component](https://www.tektutorialshub.com/angular/angular-passing-data-child-component/) tutorial, we built a counter in the parent component. We assigned the initial value to the counter and added increment/decrement methods. In the child Component, we used the @Input decorator to bind count property to the parent component. Whenever parent count is changed in the parent the child component is updated and displayed the count.

In this tutorial, we will move the counter to the child component. We will raise an event in the child component whenever the count is increased or decreased. We then bind to that event in the parent component and display the count in the parent component.

Download the source code for this from the [GitHub](https://github.com/tekTutorialsHub/Angular-Advanced-Components) from the folder inputdecorator. The The final code is available in outputdecorator folder.

#### Child Component

Open the child.component.ts and copy the following code

|  |  |
| --- | --- |
|  | import { Component, Input, Output, EventEmitter  } from '@angular/core';    @Component({      selector: 'child-component',      template: `<h2>Child Component</h2>                 <button (click)="increment()">Increment</button>                 <button (click)="decrement()">decrement</button>                 current count is {{ count }}      `  })  export class ChildComponent {      @Input() count: number;        @Output() countChanged: EventEmitter<number> =   new EventEmitter();        increment() {          this.count++;          this.countChanged.emit(this.count);        }      decrement() {          this.count--;          this.countChanged.emit(this.count);      }  } |

Now, let us look at the code in detail

First, as usual, we need to import output & EventEmitter from @angular/core

|  |  |
| --- | --- |
| 1  2  3 | import { Component, Input, Output, EventEmitter } from '@angular/core'; |

In the inline template, we have two buttons increment and decrement.  We also displaying the current count

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | @Component({      selector: 'child-component',      template: `<h2>Child Component</h2>                 <button (click)="increment()">Increment</button>                 <button (click)="decrement()">decrement</button>                 current count is {{ count }}      `  }) |

In the child component, define the countChanged event of type [EventEmiiter](https://angular.io/api/core/EventEmitter).  Decorate the property with @Output decorator to make it accessible from the parent component

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

Finally, we raise the event in increment & decrement methods using emit

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | increment() {          this.count++;          this.countChanged.emit(this.count);        }      decrement() {          this.count--;          this.countChanged.emit(this.count);      } |

#### Parent Component

In the parent component , we need to listen to the “countChanged” event

The “countChanged” event is enclosed in Parentheses. It is then assigned to the method “countChangedHandler” in the component class. The syntax is similar to [Event Binding](https://www.tektutorialshub.com/angular/angular-data-binding/#Event-Binding)

|  |  |
| --- | --- |
|  | <child-component [count]=ClickCounter (countChanged)="countChangedHandler($event)"></child-component>` |

The countChangedHandler($event) method accepts the $event argument. The data associated with event is now available to in the $event property

Our CountChangedHandler is as follows. It just updates the clickCounter and also logs the count to console

|  |  |
| --- | --- |
|  | countChangedHandler(count: number) {      this.ClickCounter = count;      console.log(count);    } |

The complete code is as follows

|  |  |
| --- | --- |
|  | import { Component} from '@angular/core';    @Component({    selector: 'app-root',    template: `          <h1>Welcome to {{title}}!</h1>          <p> current count is {{ClickCounter}} </p>          <child-component [count]=Counter (countChanged)="countChangedHandler($event)"></child-component>` ,    styleUrls: ['./app.component.css']  })  export class AppComponent {    title = 'Component Interaction';    Counter = 5;      countChangedHandler(count: number) {      this.Counter = count;      console.log(count);    }  } |

Run the code. Whenever the increment/decrement buttons clicked, The child raises the event. The Parent component gets notified of the this and updates the counter with the latest value.

## Parent uses local variable to access the Child in Template

Parent Template can access the child component properties and methods by creating the template reference variable

### Child Component

Let us update the child component

|  |  |
| --- | --- |
|  | **import { Component} from '@angular/core';**    **@Component({**  **selector: 'child-component',**  **template: `<h2>Child Component</h2>**  **current count is {{ count }}**  **`**  **})**  **export class ChildComponent {**  **count = 0;**    **increment() {**  **this.count++;**  **}**  **decrement() {**  **this.count--;**  **}**  **}** |

We have removed the input, output & eventemiitter.

Our component is now have property count and two methods to increment and decrement it

### Parent component

|  |  |
| --- | --- |
|  | **import { Component} from '@angular/core';**    **@Component({**  **selector: 'app-root',**  **template: `**  **<h1>{{title}}!</h1>**  **<p> current count is {{child.count}} </p>**  **<button (click)="child.increment()">Increment</button>**  **<button (click)="child.decrement()">decrement</button>**  **<child-component #child></child-component>` ,**  **styleUrls: ['./app.component.css']**  **})**  **export class AppComponent {**  **title = 'Parent interacts with child via local variable';**  **}** |

We have created a local variable, #child, on the tag <child-component>. The “child” is called template reference variable, which now represents the child component

The Template Reference variable is created, when you use #<varibaleName> and attach it to a DOM element. You can then, use the variable to reference the DOM element in your Template

|  |  |
| --- | --- |
| 1  2  3 | <child-component #child></child-component>` , |

Now you can use the local variable elsewhere in the template to refer to the child component methods and properties as shown below

|  |  |
| --- | --- |
|  | <p> current count is {{child.count}} </p>          <button (click)="child.increment()">Increment</button>          <button (click)="child.decrement()">decrement</button> |

The code above wires child components increment & decrement methods from the parent component

The local variable approach is simple and easy. But it is limited because the parent-child wiring must be done entirely within the parent template. The parent component itself has no access to the child.

You can’t use the local variable technique if an instance of the parent component class must read or write child component values or must call child component methods.

## Parent uses a @ViewChild() to get reference to the Child Component

**Injecting an instance of the child component into the parent as a @ViewChild is the another technique used by the parent to access the property and method of the child component**

**The @ViewChild decorator takes the name of the component/directive as its input. It is then used to decorate a property. The Angular then injects the reference of the component to the Property**

**For Example**

**In the Parent component, declare a property child which is of type ChildComponent**

|  |  |
| --- | --- |
|  | **child: ChildComponent;** |

**Next, decorate it with @ViewChild decorator passing it the name of the component to inject**

|  |  |
| --- | --- |
|  | **@ViewChild(ChildComponent) child: ChildComponent;** |

**Now, when angular creates the child component, the reference to the child component is assigned to the child property.**

**We now update the code from previous section**

### Child Component

**There is no change in the child component**

### Parent Component

**In the parent component, we need to import the viewChild Decorator. We also need to import the child component**

|  |  |
| --- | --- |
|  | **import { Component, ViewChild } from '@angular/core';**  **import { ChildComponent } from './child.component';** |

**Next, create a property child which is an instance of type ChildComponent. Apply the viewChild Decorator on the child component as shown below**

|  |  |
| --- | --- |
|  | **@ViewChild(ChildComponent) child: ChildComponent;** |

**Finally, add increment and decrement method, which invokes the methods in the child component**

|  |  |
| --- | --- |
|  | **increment() {**  **this.child.increment();**  **}**    **decrement() {**  **this.child.decrement();**  **}** |

**Now, the parent can access the properties and methods of child component**

**And in the template make necessary changes**

|  |  |
| --- | --- |
| **7** | **<h1>{{title}}</h1>**  **<p> current count is {{child.count}} </p>**  **<button (click)="increment()">Increment</button>**  **<button (click)="decrement()">decrement</button>**  **<child-component></child-component>`** |

**The complete app.component.ts is as follows**

|  |  |
| --- | --- |
|  | **import { Component, ViewChild } from '@angular/core';**  **import { ChildComponent } from './child.component';**    **@Component({**  **selector: 'app-root',**  **template: `**  **<h1>{{title}}</h1>**  **<p> current count is {{child.count}} </p>**  **<button (click)="increment()">Increment</button>**  **<button (click)="decrement()">decrement</button>**  **<child-component></child-component>` ,**  **styleUrls: ['./app.component.css']**  **})**  **export class AppComponent {**  **title = 'Parent calls an @ViewChild()';**    **@ViewChild(ChildComponent) child: ChildComponent;**    **increment() {**  **this.child.increment();**  **}**    **decrement() {**  **this.child.decrement();**  **}**  **}** |

## **Conclusion**

**In this tutorial, we looked at how the parent can communicate with the child component. The Parent can subscribe to the events of the child component. It can use the Template local variable to access the properties and methods. We can also use @ViewChild decorator to inject the child component instance to the parent**

13