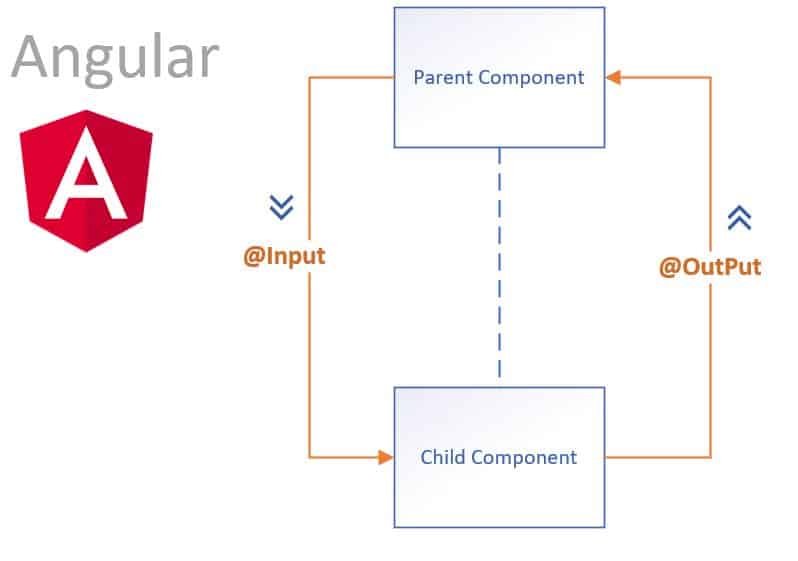
**Angular @Output**

The Decorator that marks a class field as an output property and supplies configuration metadata. The DOM property bound to an output property is automatically updated during change detection.

You can supply the optional name to use in the templates when a component is instantiated that maps to the name of the bound property. By default, the original name of the required property is used for output binding.

Now, let’s understand this with an example. But, first, let’s create a new Angular project.



**Step 1: Install the Angular Project via AngularCLI.**

First, we need to install Angular CLI globally in our system by typing the following command.

npm install -g @angular/cli

Now, fire the following command to create a project.

ng **new** **inout**

**Step 2: Create parent and child components.**

Go to the terminal and type the following command.

ng g **c** parent

ng g **c** child

So, it will create an individual folder. Type the following command to start the Angular development server.

ng serve *--open*

It will open up the browser at the **port: 4200**.

Right now, only the **app.component.ts**component is rendered in the browser. If we want to render our parent component, we need to include it in an **app.component.html**file as an HTML tag.

<div style="text-align:center">

<h1>

Welcome to {{ title }}!

</h1>

<app-parent></app-parent>

</div>

Now, if you see in the browser, you can see the parent component renders. “**parent works!!** “

**Step 3: Define HTML for parent component.**

Write the following code in the **parent.component.html**file.

<h3>Parent **Component**</h3>

<**label**>Bitcoin price</**label**>

<input **type**="text" />

<p>Value **of** child **component** **is**: </p>

First, we pass the data from the parent component to the child component. Here is the scenario, when the user types the bitcoin price in the text box, we can see its worth in the child component.

The same scenario applies to the child component. When the child component starts entering the price, it will display in the parent component.

**Step 4: Define HTML for child components.**

Write the following code in the **child.component.html**file.

<h3>Child **Component**</h3>

<**label**>Child **Component**</**label**>

<input **type**="text" />

<p>Value **of** parent **component** **is**: </p>

As we know, this is a child component, so we need to include the **<app-child>** tag into the parent component. So our parent component HTML looks like this.

<h3>Parent Component</h3>

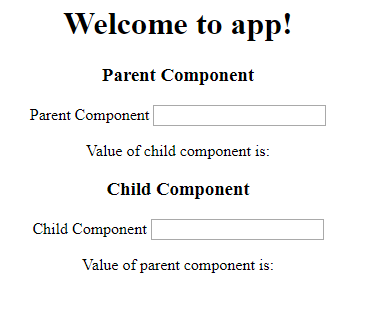
<label>Parent Component</label>

<input type="text" />

<p>Value of child component is: </p>

<app-child></app-child>

So, our application looks like this.



**Step 5: Use Input to display parent component value**

Create a reference to the input text of the parent component. To edit the following lines in the **parent,component.html**file.

<**input** type="text" **#pcomponent** (keyup)="0"/>

<app-child [PData]="pcomponent.value"></app-child>

First, I have defined the reference for the input tag and then set the event listener. Then, when a user types something in the textbox, it will pass the value as a property to the child component.

The child component is ready to receive the property via the **@Input Decorator**. So this is the first use case of Inputs in Angular.

File **child.component.ts**file looks like this.

*// child.component.ts*

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

**@Component**({

selector: 'app-child',

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

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

})

**export** **class** ChildComponent **implements** OnInit {

**@Input**() PData: number;

**constructor**() { }

ngOnInit() {

}

}

You can see this component’s property is PData, which is the same property we have written in the **parent.component.html**file.

Finally, our **child.component.html**file looks like this. We need to add interpolation to display the parent data in the child component.

<h3>Child Component</h3>

<label>Child Component</label>

<input type="text" />

<p>Value of parent component is: {{ PData }}</p>

Now, if you type the parent text box, then its value print in the child component. Thus, all is done through the parent to child node via input property.

**Step 6: Pass value from child to parent component.**

Passing the data from the child component to the parent component is a little bit tricky. In this scenario, the child component does not have any reference to the parent component.

So, in this case, we need to emit an event from the child component, and the parent component will listen to it and receive the data via event and display it.

First, create a reference to the Input in the child component and attach an event listener to it.

<h3>Child Component</h3>

<label>Child Component</label>

<input type="text" #ccomponent (keyup)="onChange(ccomponent.value)"/>

<p>Value of parent component is: {{ PData }}</p>

Write the onChange function in the **child.component.ts**file.

*// child.component.ts*

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

**@Component**({

selector: 'app-child',

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

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

})

**export** **class** ChildComponent **implements** OnInit {

**@Input**() PData: number;

**@Output**() childEvent = **new** EventEmitter();

**constructor**() { }

onChange(value) {

**this**.childEvent.emit(value);

}

ngOnInit() {

}

}

When the user types anything in the textbox of the child component, it will start emitting the value from the child component. So we just need to listen to that event emitter and display the passed value in the parent component.

Use an event binding in the **parent.component.html**file and listen for the event emitter.

<**app**-child [PData]="pcomponent.value" (childEvent)="CData=$event"></**app**-child>

We need to define CData into the **parent.component.ts**file.

*// parent.component.ts*

**public** CData: number;

Finally, by interpolation, we can display its value in the **parent.component.html** file.

// parent.component.html

<h3>Parent Component</h3>

<label>Parent Component</label>

<input type="text" #pcomponent (keyup)="0"/>

<p>Value of child component is: {{ CData }}</p>

<app-child [PData]="pcomponent.value" (childEvent)="CData=$event"></app-child>

I am writing the following four files if you find any confusion throughout this tutorial.

*// parent.component.ts*

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

**@Component**({

selector: 'app-parent',

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

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

})

**export** **class** ParentComponent **implements** OnInit {

**public** CData: number;

**constructor**() { }

ngOnInit() {

}

}

*// child.component.ts*

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

**@Component**({

selector: 'app-child',

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

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

})

**export** **class** ChildComponent **implements** OnInit {

**@Input**() PData: number;

**@Output**() childEvent = **new** EventEmitter();

**constructor**() { }

onChange(value) {

**this**.childEvent.emit(value);

}

ngOnInit() {

}

}

// child.component.html

<h3>Child Component</h3>

<label>Child Component</label>

<input type="text" #ccomponent (keyup) = "onChange(ccomponent.value)"/>

<p>Value of parent component is: {{ PData }}</p>

Finally, our **Angular Input Output Example** is over.