


## Using ViewChild in Angular to Access a Child Component, Directive or DOM Element

Want to get access to a child component, directive or a DOM element from a parent component class? It's easy to do with the ViewChild decorator. ViewChild returns the first element that matches a given component, directive or template reference selector. In cases where you'd want to access multiple children, you'd use ViewChildren instead.

The nice thing too is that, if the reference changes to a new element dynamically, ViewChild will take care of updating its reference automatically.

 Alligator.io recommends [1](#)

### Directives

Let's say we have a BaconDirective like this:

```
bacon.directive.ts
import { Directive, ElementRef, Renderer2 } from '@angular/core';

@Directive({ selector: '[appBacon]' })
export class BaconDirective {
  ingredient = 'mayo';

  constructor(elem: ElementRef, renderer: Renderer2) {
    let bacon = renderer.createText('🥓🥓🥓 ');
    renderer.appendChild(elem.nativeElement, bacon);
  }
}
```

And we use it in our component template like this, to add some bacon to our sandwich:

```
<span appBacon>sandwich!</span>
```

The result: 🥓🥓🥓 sandwich!

---

We can now access the directive with ViewChild using something like the following snippet in our component class.

For this example, we'll access the ingredient instance variable of our directive and set an extraIngredient instance variable with its value:

```
app.component.ts
import { Component, ViewChild, AfterViewInit } from '@angular/core';

import { BaconDirective } from './bacon.directive'

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements AfterViewInit {
  extraIngredient: string;

  @ViewChild(BaconDirective)
  set appBacon(directive: BaconDirective) {
    this.extraIngredient = directive.ingredient;
  };

  ngAfterViewInit() {
```

```

    console.log(this.extraIngredient); // mayo
  }
}

```

We used a setter here to set our extraIngredient variable. Notice that we wait for the AfterViewInit [lifecycle hook](#) to access our variable, as this is when child components and directives become available.

## DOM Elements

We can access native DOM elements that have a template reference variable. Let's say we have this in our template with the someInput reference variable:

```
<input #someInput placeholder="Your favorite pizza topping">
```

We can access the input itself with ViewChild like this:

```

import { Component,
  ViewChild,
  AfterViewInit,
  ElementRef } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements AfterViewInit {
  @ViewChild('someInput') someInput: ElementRef;

  ngAfterViewInit() {
    this.someInput.nativeElement.value = "Anchovies! 🍷🍷";
  }
}

```

And the value of our input will be set to Anchovies! 🍷🍷 when ngAfterViewInit fires.

## Child Components

It's just as easy to access a child component and call methods or access instance variables that are available on the child. Let's say we have a child component with a whoAmI method like this:

```

whoAmI() {
  return '👶 I am a child!!';
}

```

We can then call that method from within our parent component class with ViewChild like this:

```

                                                                    app.component.ts
import { Component,
  ViewChild,
  AfterViewInit } from '@angular/core';

import { ChildComponent } from './child.component';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements AfterViewInit {
  @ViewChild(ChildComponent) child: ChildComponent;

  ngAfterViewInit() {
    console.log(this.child.whoAmI()); // 👶 I am a child!
  }
}

```