Observables

Angular makes use of observables as an interface to handle a variety of common asynchronous operations. For example:

* You can define [custom events](https://angular.io/guide/event-binding#custom-events-with-eventemitter) that send observable output data from a child to a parent component.
* The HTTP module uses observables to handle AJAX requests and responses.
* The Router and Forms modules use observables to listen for and respond to user-input events

Observables are declarative—that is, you define a function for publishing values, but it is not executed until a consumer subscribes to it. The subscribed consumer then receives notifications until the function completes, or until they unsubscribe.

## Basic usage and terms

As a publisher, you create an Observable instance that defines a subscriber function. This is the function that is executed when a consumer calls the subscribe() method. The subscriber function defines how to obtain or generate values or messages to be published.

To execute the observable we need to call its subscribe() method, passing an observer.

The subscribe() call returns a Subscription object that has an unsubscribe() method, which you call to stop receiving notifications.

## Observers

## A handler for receiving observable notifications implements the Observer interface. It is an object that defines callback methods to handle the three types of notifications that an observable can send:

| NOTIFICATION TYPE | DESCRIPTION |
| --- | --- |
| next | Required. A handler for each delivered value. Called zero or more times after execution starts. |
| error | Optional. A handler for an error notification. An error halts execution of the observable instance. |
| complete | Optional. A handler for the execution-complete notification. Delayed values can continue to be delivered to the next handler after execution is complete. |

Example

component.ts

**myObservable = new Observable(observer =>{**

**setTimeout(() =>{**

**observer.next('Started Reading')**

**}, 2000);**

**setTimeout(() =>{**

**observer.next('Still Reading')**

**}, 4000);**

**setTimeout(() =>{**

**observer.next('Please Read Fast')**

**}, 6000);**

**setTimeout(() =>{**

**observer.next('Finished Reading')**

**}, 8000);**

**})**

ngOnInit(): void {

this.myObservable.subscribe(val =>{

this.bookStatus = val;

});

}

\*\*\*\*\*Error\*\*\*\*\*\*\*\*\*\*\*

setTimeout(() =>{

observer.error(new Error('Pages missing in the book can\'t read any more' ))

}, 6000);

ngOnInit(): void {

this.myObservable.subscribe(val =>{

this.bookStatus = val;

},

(error)=>{

alert(error.message);

});

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Complete \*\*\*\*\*\*\*\*\*\*\*\*

setTimeout(() =>{

observer.complete()

}, 6000);

ngOnInit(): void {

this.myObservable.subscribe(val =>{

this.bookStatus = val;

},

(error)=>{

alert(error.message);

},

() => {

alert('Read the book earlier')

});

}

Multiple Subscriptions

 bookStatus: any;

  bookStatus1: any;

  data: **Observable**<any> | undefined

**ngOnInit**(): void {

    this.data = new **Observable**(observer =>{

**setTimeout**(()=> {

        observer.**next**('Started Reading')

      }, 2000);

**setTimeout**(()=> {

        observer.**next**('Still Reading')

      }, 6000);

**setTimeout**(()=> {

        observer.**next**('Finished Reading')

      }, 10000);

    } );

    this.data.**subscribe**(val =>{

      this.bookStatus = val;

    });

    this.data.**subscribe**(val2 =>{

      this.bookStatus1 = val2;

    });

  }

Subject

Subject is a special type of Observable in RxJs Library in which we can send our data to other components or services. A Subject is like an Observable but can multicast to many observers which means subject is at the same time an Observable and an Observer.

* A Subject is a Special type of Observable that allows value to be multicasted to many Observers.
* Subject are like event emitters.
* No Initial Value

subject =**new** Subject<datatype>();

Behavior Subject

Behavior Subject is similar to subject but only difference is that we can set the initial value .

A variant of subject that requires initial value.

**const** subject =**new** BehaviorSubject(0); //0 is the initial value.

**\*\*\*\*\*\*\*\*\*Example for Subject\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**ng g c components/component1**

**ng g c components/component2**

**ng g c components/component3**

**ng g s services/datasharing**

///////// component1.component.ts

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

import { DataSharingService } from '../datasharing.service';

@Component({

  selector: 'app-component1',

  templateUrl: './component1.component.html',

  styleUrls: ['./component1.component.css']

})

export class Component1Component implements OnInit {

  Component1Data: any = '';

  constructor(private DataSharing: DataSharingService) {

    this.DataSharing.SharingData.subscribe((res: any) => {

      this.Component1Data = res;

    })

  }

  ngOnInit(){}

  onSubmit(data) {

    this.DataSharing.SharingData.next(data.value);

  }

}

/////// component1.component.html

<div style="background-color: aliceblue;" class="card">

  <div class="card-body">

    <h5 class="card-title">Component 1</h5>

    <input name="comp1" #comp1 type="text" />

    <button (click)="onSubmit(comp1)">Submit</button>

    <p class="card-text">{{Component1Data}}</p>

  </div>

</div>

//////// component2.component.ts

Component2Data: any = '';

  constructor(private DataSharing: DataSharingService) {

    this.DataSharing.SharingData.subscribe((res: any) => {

      this.Component2Data = res;

    })

  }

  onSubmit(data) {

    this.DataSharing.SharingData.next(data.value);

  }

///////////////// component2.component.html

<div style="background-color:antiquewhite" class="card">

  <div class="card-body">

    <h5 class="card-title">Component 2</h5>

    <input name="comp2" #comp2 type="text" />

    <button (click)="onSubmit(comp2)">Submit</button>

    <p class="card-text">{{Component2Data}}</p>

  </div>

</div>

///////////////////// component3.component.ts

  Component3Data: any = '';

  constructor(private DataSharing: DataSharingService) {

    this.DataSharing.SharingData.subscribe((res: any) => {

      this.Component3Data = res;

    })

  }

  onSubmit(data) {

    this.DataSharing.SharingData.next(data.value);

  }

/////////////////// component3.comopntnt.html

<div style="background-color:burlywood" class="card">

  <div class="card-body">

    <h5 class="card-title">Component 3</h5>

    <input name="comp3" #comp3 type="text" />

    <button (click)="onSubmit(comp3)">Submit</button>

    <p class="card-text">{{Component3Data}}</p>

  </div>

</div>

///////////////////////////// datasharing.service.ts

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

import { Subject } from 'rxjs';

@Injectable()

export class DataSharingService {

SharingData = new Subject();

  constructor() { }

}