
Table of Contents

Learn RxJS

Introduction	1.1
Operators	1.2
Combination	1.2.1
combineAll	1.2.1.1
combineLatest	1.2.1.2
concat	1.2.1.3
concatAll	1.2.1.4
endWith	1.2.1.5
forkJoin	1.2.1.6
merge	1.2.1.7
mergeAll	1.2.1.8
pairwise	1.2.1.9
race	1.2.1.10
startWith	1.2.1.11
withLatestFrom	1.2.1.12
zip	1.2.1.13
Conditional	1.2.2
defaultIfEmpty	1.2.2.1
every	1.2.2.2
iif	1.2.2.3
sequenceEqual	1.2.2.4
Creation	1.2.3
ajax	1.2.3.1
create	1.2.3.2
defer	1.2.3.3

empty	1.2.3.4
from	1.2.3.5
fromEvent	1.2.3.6
generate	1.2.3.7
interval	1.2.3.8
of	1.2.3.9
range	1.2.3.10
throw	1.2.3.11
timer	1.2.3.12
Error Handling	1.2.4
catch / catchError	1.2.4.1
retry	1.2.4.2
retryWhen	1.2.4.3
Multicasting	1.2.5
publish	1.2.5.1
multicast	1.2.5.2
share	1.2.5.3
shareReplay	1.2.5.4
Filtering	1.2.6
audit	1.2.6.1
auditTime	1.2.6.2
debounce	1.2.6.3
debounceTime	1.2.6.4
distinct	1.2.6.5
distinctUntilChanged	1.2.6.6
distinctUntilKeyChanged	1.2.6.7
filter	1.2.6.8
find	1.2.6.9
first	1.2.6.10
ignoreElements	1.2.6.11

last	1.2.6.12
sample	1.2.6.13
single	1.2.6.14
skip	1.2.6.15
skipUntil	1.2.6.16
skipWhile	1.2.6.17
take	1.2.6.18
takeLast	1.2.6.19
takeUntil	1.2.6.20
takeWhile	1.2.6.21
throttle	1.2.6.22
throttleTime	1.2.6.23
Transformation	1.2.7
buffer	1.2.7.1
bufferCount	1.2.7.2
bufferTime	1.2.7.3
bufferToggle	1.2.7.4
bufferWhen	1.2.7.5
concatMap	1.2.7.6
concatMapTo	1.2.7.7
exhaustMap	1.2.7.8
expand	1.2.7.9
groupBy	1.2.7.10
map	1.2.7.11
mapTo	1.2.7.12
mergeMap / flatMap	1.2.7.13
mergeScan	1.2.7.14
partition	1.2.7.15
pluck	1.2.7.16
reduce	1.2.7.17

scan	1.2.7.18
switchMap	1.2.7.19
switchMapTo	1.2.7.20
toArray	1.2.7.21
window	1.2.7.22
windowCount	1.2.7.23
windowTime	1.2.7.24
windowToggle	1.2.7.25
windowWhen	1.2.7.26
Utility	1.2.8
tap / do	1.2.8.1
delay	1.2.8.2
delayWhen	1.2.8.3
dematerialize	1.2.8.4
finalize / finally	1.2.8.5
let	1.2.8.6
repeat	1.2.8.7
timeInterval	1.2.8.8
timeout	1.2.8.9
timeoutWith	1.2.8.10
toPromise	1.2.8.11
Full Listing	1.2.9
Subjects	1.3
AsyncSubject	1.3.1
BehaviorSubject	1.3.2
ReplaySubject	1.3.3
Subject	1.3.4
Recipes	1.4
Alphabet Invasion Game	1.4.1
Battleship Game	1.4.2

Breakout Game	1.4.3
Car Racing Game	1.4.4
Catch The Dot Game	1.4.5
Click Ninja Game	1.4.6
Flappy Bird Game	1.4.7
Game Loop	1.4.8
Horizontal Scroll Indicator	1.4.9
Http Polling	1.4.10
Lockscreen	1.4.11
Matrix Digital Rain	1.4.12
Memory Game	1.4.13
Mine Sweeper Game	1.4.14
Platform Jumper Game	1.4.15
Progress Bar	1.4.16
Save Indicator	1.4.17
Smart Counter	1.4.18
Space Invaders Game	1.4.19
Stop Watch	1.4.20
Swipe To Refresh	1.4.21
Tank Battle Game	1.4.22
Tetris Game	1.4.23
Type Ahead	1.4.24
Concepts	1.5
RxJS Primer	1.5.1
RxJS v5 -> v6 Upgrade	1.5.2
Time based operators comparison	1.5.3
Understanding Operator Imports	1.5.4

Learn RxJS

Clear examples, explanations, and resources for RxJS.

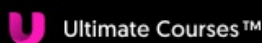
By [@btroncone](#)

Introduction

[RxJS](#) is one of the hottest libraries in web development today. Offering a powerful, functional approach for dealing with events and with integration points into a growing number of frameworks, libraries, and utilities, the case for learning Rx has never been more appealing. Couple this with the ability to utilize your knowledge across [nearly any language](#), having a solid grasp on reactive programming and what it can offer seems like a no-brainer.

But...

Learning RxJS and reactive programming is [hard](#). There's the multitude of concepts, large API surface, and fundamental shift in mindset from an [imperative to declarative style](#). This site focuses on making these concepts approachable, the examples clear and easy to explore, and features references throughout to the best RxJS related material on the web. The goal is to supplement the [official docs](#) and pre-existing learning material while offering a new, fresh perspective to clear any hurdles and tackle the pain points. Learning Rx may be difficult but it is certainly worth the effort!



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Brand New to RxJS?

Start getting familiar with all the key concepts needed to be productive with our [RxJS Primer](#)!

Content

Operators

Operators are the horse-power behind observables, providing an elegant, declarative solution to complex asynchronous tasks. This section contains all [RxJS operators](#), included with clear, executable examples. Links to additional resources and recipes for each operator are also provided, when applicable.

Operator Categories

- [Combination](#)
- [Conditional](#)
- [Creation](#)
- [Error Handling](#)
- [Multicasting](#)
- [Filtering](#)
- [Transformation](#)
- [Utility](#)

OR...

[Complete listing in alphabetical order](#)

Understanding Subjects

A Subject is a special type of Observable which shares a single execution path among observers.

- [Overview](#)
- [AsyncSubject](#)
- [BehaviorSubject](#)
- [ReplaySubject](#)

- [Subject](#)

Concepts

Without a solid base knowledge of how Observables work behind the scenes, it's easy for much of RxJS to feel like 'magic'. This section helps solidify the major concepts needed to feel comfortable with reactive programming and Observables.

- [RxJS Primer](#)
- [RxJS v5 -> v6 Upgrade](#)
- [Time based operators comparison](#)
- [Understanding Operator Imports](#)

Recipes

Recipes for common use-cases and interesting solutions with RxJS.

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Catch The Dot Game](#)
- [Click Ninja Game](#)
- [Flappy Bird Game](#)
- [Game Loop](#)
- [Horizontal Scroll Indicator](#)
- [HTTP Polling](#)
- [Lockscreen](#)
- [Matrix Digital Rain](#)
- [Memory Game](#)
- [Mine Sweeper Game](#)
- [Platform Jumper Game](#)
- [Progress Bar](#)
- [Save Indicator](#)
- [Smart Counter](#)
- [Stop Watch](#)
- [Space Invaders Game](#)
- [Swipe To Refresh](#)

- [Tank Battle Game](#)
- [Tetris Game](#)
- [Type Ahead](#)

Introductory Resources

New to RxJS and reactive programming? In addition to the content found on this site, these excellent resources will help jump start your learning experience!

Conferences

- [RxJS Live](#) - RxJS specific conference, Las Vegas

Reading

- [RxJS Introduction](#) - Official Docs
- [The Introduction to Reactive Programming You've Been Missing](#) - André Staltz
- [RxJS: Observables, Observers and Operators Introduction](#) - Todd Motto

Videos

- [RxJS Basics](#) 📺 - Brian Troncone
- [Asynchronous Programming: The End of The Loop](#) - Jafar Husain
- [What is RxJS?](#) - Ben Lesh
- [Creating Observable from Scratch](#) - Ben Lesh
- [Introduction to RxJS Marble Testing](#) 📺 - Brian Troncone
- [Introduction to Reactive Programming](#) 📺 - André Staltz
- [Reactive Programming using Observables](#) - Jeremy Lund

Exercises

- [Functional Programming in JavaScript](#) - Jafar Husain

Tools

- [Rx Marbles - Interactive diagrams of Rx Observables](#) - André Staltz
- [Rx Visualizer - Animated playground for Rx Observables](#) - Misha Moroshko

- [Reactive.how - Animated cards to learn Reactive Programming](#) - Cédric Soulas
- [Rx Visualization - Visualizes programming with RxJS](#) - Mojtaba Zarei

Interested in RxJS 4? Check out [Denis Stoyanov's](#) excellent [eBook](#)!

Translations

- [简体中文](#)

A Note On References

All references included in this GitBook are resources, both free and paid, that helped me tremendously while learning RxJS. If you come across an article or video that you think should be included, please use the *edit this page* link in the top menu and submit a pull request. Your feedback is appreciated!

RxJS Operators By Example

A complete list of RxJS operators with clear explanations, relevant resources, and executable examples.

[Prefer a complete list in alphabetical order?](#)

Contents (By Operator Type)

- **Combination**
 - [combineAll](#)
 - [combineLatest](#) ★
 - [concat](#) ★
 - [concatAll](#)
 - [endWith](#)
 - [forkJoin](#)
 - [merge](#) ★
 - [mergeAll](#)
 - [pairwise](#)
 - [race](#)
 - [startWith](#) ★
 - [withLatestFrom](#) ★
 - [zip](#)
- **Conditional**
 - [defaultIfEmpty](#)
 - [every](#)
 - [iif](#)
 - [sequenceequal](#)
- **Creation**
 - [ajax](#) ★
 - [create](#)
 - [defer](#)
 - [empty](#)
 - [from](#) ★
 - [fromEvent](#)




- generate
- interval
- of ★
- range
- throw
- timer
- Error Handling
 - catch / catchError ★
 - retry
 - retryWhen
- Filtering
 - audit
 - auditTime
 - debounce
 - debounceTime ★
 - distinct
 - distinctUntilChanged ★
 - distinctUntilKeyChanged
 - filter ★
 - find
 - first
 - ignoreElements
 - last
 - sample
 - single
 - skip
 - skipUntil
 - skipWhile
 - take ★
 - takeLast
 - takeUntil ★
 - takeWhile
 - throttle
 - throttleTime
- Multicasting
 - multicast

- publish
- share ★
- shareReplay ★
- Transformation
 - buffer
 - bufferCount
 - bufferTime ★
 - bufferToggle
 - bufferWhen
 - concatMap ★
 - concatMapTo
 - expand
 - exhaustMap
 - groupBy
 - map ★
 - mapTo
 - mergeMap / flatMap ★
 - mergeScan
 - partition
 - pluck
 - reduce
 - scan ★
 - switchMap ★
 - switchMapTo
 - toArray
 - window
 - windowCount
 - windowTime
 - windowToggle
 - windowWhen
- Utility
 - tap / do ★
 - delay
 - delayWhen
 - dematerialize
 - finalize / finally

- [let](#)
- [repeat](#)
- [repeatWhen](#)
- [timeInterval](#)
- [timeout](#)
- [timeoutWith](#)
- [toPromise](#)

★ - *commonly used*

Additional Resources

- [What Are Operators?](#)  - Official Docs
- [What Operators Are](#)   - André Staltz

Combination Operators

The combination operators allow the joining of information from multiple observables. Order, time, and structure of emitted values is the primary variation among these operators.

Contents

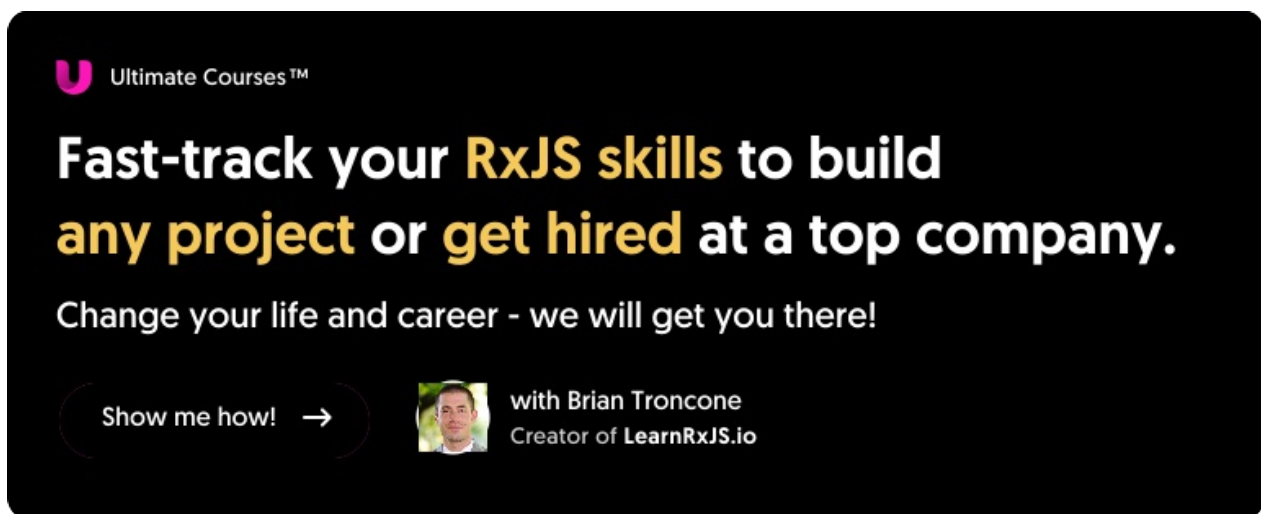
- [combineAll](#)
- [combineLatest](#) ★
- [concat](#) ★
- [concatAll](#)
- [endWith](#)
- [forkJoin](#)
- [merge](#) ★
- [mergeAll](#)
- [pairwise](#)
- [race](#)
- [startWith](#) ★
- [withLatestFrom](#) ★
- [zip](#)

★ - *commonly used*

combineAll

signature: `combineAll(project: function): Observable`

When source observable completes use `combineLatest` with collected observables.




U Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

([example tests](#))

Example 1: Mapping to inner interval observable

([StackBlitz](#))


```
import { take, map, combineAll } from 'rxjs/operators';
import { interval } from 'rxjs';

// emit every 1s, take 2
const source$ = interval(1000).pipe(take(2));
// map each emitted value from source to interval observable that takes 5 values
const example$ = source$.pipe(
  map(val =>
    interval(1000).pipe(
      map(i => `Result (${val}): ${i}`),
      take(5)
    )
  )
);
/*
  2 values from source will map to 2 (inner) interval observables that emit every 1s.
  combineAll uses combineLatest strategy, emitting the last value from each
  whenever either observable emits a value
*/
example$
  .pipe(combineAll())
  /*
    output:
    ["Result (0): 0", "Result (1): 0"]
    ["Result (0): 1", "Result (1): 0"]
    ["Result (0): 1", "Result (1): 1"]
    ["Result (0): 2", "Result (1): 1"]
    ["Result (0): 2", "Result (1): 2"]
    ["Result (0): 3", "Result (1): 2"]
    ["Result (0): 3", "Result (1): 3"]
    ["Result (0): 4", "Result (1): 3"]
    ["Result (0): 4", "Result (1): 4"]
  */
  .subscribe(console.log);
```

Additional Resources

- [combineAll](#)  - Official docs



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/combineAll.ts>

combineLatest

signature: `combineLatest(observables: ...Observable,
project: function): Observable`

When any observable emits a value, emit the last emitted value from each.

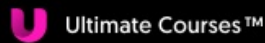
💡 `combineAll` can be used to apply `combineLatest` to emitted observables when a source completes!

Why use `combineLatest` ?

This operator is best used when you have multiple, long-lived observables that rely on each other for some calculation or determination. Basic examples of this can be seen in [example three](#), where events from multiple buttons are being combined to produce a count of each and an overall total, or a [calculation of BMI](#) from the RxJS documentation.

Be aware that `combineLatest` **will not emit an initial value until each observable emits at least one value**. This is the same behavior as `withLatestFrom` and can be a *gotcha* as there will be no output and no error but one (or more) of your inner observables is likely not functioning as intended, or a subscription is late.

Lastly, if you are working with observables that only emit one value, or you only require the last value of each before completion, `forkJoin` is likely a better option.



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Combining observables emitting at 3 intervals

([StackBlitz](#))

```
// RxJS v6+
import { timer, combineLatest } from 'rxjs';

// timerOne emits first value at 1s, then once every 4s
const timerOne$ = timer(1000, 4000);
// timerTwo emits first value at 2s, then once every 4s
const timerTwo$ = timer(2000, 4000);
// timerThree emits first value at 3s, then once every 4s
const timerThree$ = timer(3000, 4000);

// when one timer emits, emit the latest values from each timer
// as an array
combineLatest(timerOne$, timerTwo$, timerThree$).subscribe(
  ([timerValOne, timerValTwo, timerValThree]) => {
    /*
      Example:
      timerOne first tick: 'Timer One Latest: 1, Timer Two Latest:
0, Timer Three Latest: 0
      timerTwo first tick: 'Timer One Latest: 1, Timer Two Latest:
1, Timer Three Latest: 0
      timerThree first tick: 'Timer One Latest: 1, Timer Two Latest:
1, Timer Three Latest: 1
    */
    console.log(
      `Timer One Latest: ${timerValOne},
      Timer Two Latest: ${timerValTwo},
      Timer Three Latest: ${timerValThree}`
    );
  }
);
```

Example 2: combineLatest with projection function

([StackBlitz](#))

```
// RxJS v6+
import { timer, combineLatest } from 'rxjs';

const timerOne$ = timer(1000, 4000);
const timerTwo$ = timer(2000, 4000);
const timerThree$ = timer(3000, 4000);

combineLatest(
  timerOne$,
  timerTwo$,
  timerThree$,
  // combineLatest also takes an optional projection function
  (one, two, three) => {
    return `Timer One (Proj) Latest: ${one},
           Timer Two (Proj) Latest: ${two},
           Timer Three (Proj) Latest: ${three}`;
  }
).subscribe(console.log);
```

Example 3: Combining events from 2 buttons

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent, combineLatest } from 'rxjs';
import { mapTo, startWith, scan, tap, map } from 'rxjs/operators';

// elem refs
const redTotal = document.getElementById('red-total');
const blackTotal = document.getElementById('black-total');
const total = document.getElementById('total');

const addOneClick$ = id =>
  fromEvent(document.getElementById(id), 'click').pipe(
    // map every click to 1
    mapTo(1),
    // keep a running total
    scan((acc, curr) => acc + curr, 0),
    startWith(0)
  );

combineLatest(addOneClick$('red'), addOneClick$('black')).subscribe(
  ([red, black]: any) => {
    redTotal.innerHTML = red;
    blackTotal.innerHTML = black;
    total.innerHTML = red + black;
  }
);
```

HTML

```
<div>
  <button id="red">Red</button>
  <button id="black">Black</button>
</div>
<div>Red: <span id="red-total"></span></div>
<div>Black: <span id="black-total"></span></div>
<div>Total: <span id="total"></span></div>
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Flappy Bird Game](#)
- [Platform Jumper Game](#)
- [Tank Battle Game](#)
- [Tetris Game](#)

Additional Resources

- [combineLatest](#) 📄 - Official docs
- [Combining streams with combineLatest](#) 🖥️ 💰 📄 - John Linquist
- [Combination operator: combineLatest](#) 🖥️ 💰 📄 - André Staltz
- [Build your own combineLatest operator](#) 🖥️ - Kwinten Pisman



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/combineLatest.ts>

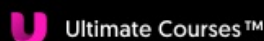
concat

signature: `concat(observables: ...*): Observable`

Subscribe to observables in order as previous completes

💡 You can think of concat like a line at a ATM, the next transaction (subscription) cannot start until the previous completes!

💡 If throughput, not order, is a primary concern, try [merge](#) instead!



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Basic concat usage with three observables

([StackBlitz](#))

```
// RxJS v6+
import { of, concat } from 'rxjs';

concat(
  of(1, 2, 3),
  // subscribed after first completes
  of(4, 5, 6),
  // subscribed after second completes
  of(7, 8, 9)
)
// log: 1, 2, 3, 4, 5, 6, 7, 8, 9
.subscribe(console.log);
```

Example 2: Display message using concat with delayed observables

([StackBlitz](#))



Get Ready!

```
// RxJS v6+
import { concat, empty } from 'rxjs';
import { delay, startWith } from 'rxjs/operators';

// elems
const userMessage = document.getElementById('message');
// helper
const delayedMessage = (message, delayedTime = 1000) => {
  return empty().pipe(
    startWith(message),
    delay(delayedTime)
  );
};

concat(
  delayedMessage('Get Ready!'),
  delayedMessage(3),
  delayedMessage(2),
  delayedMessage(1),
  delayedMessage('Go!'),
  delayedMessage('', 2000)
).subscribe((message: any) => (userMessage.innerHTML = message));
```

Example 3: (Warning!) concat with source that does not complete

([StackBlitz](#))



```
// RxJS v6+
import { interval, of, concat } from 'rxjs';

// when source never completes, any subsequent observables never
run
concat(interval(1000), of('This', 'Never', 'Runs'))
  // log: 1,2,3,4.....
  .subscribe(console.log);
```

Related Recipes

- [Battleship Game](#)
- [Save Indicator](#)

Additional Resources

- [concat](#)  - Official docs
- [Combination operator: concat, startWith](#)   - André Staltz



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/concat.ts>

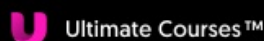
concatAll

signature: `concatAll(): Observable`

Collect observables and subscribe to next when previous completes.

⚠ Be wary of [backpressure](#) when the source emits at a faster pace than inner observables complete!

💡 In many cases you can use [concatMap](#) as a single operator instead!



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

([example tests](#))

Example 1: concatAll with observable

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { map, concatAll } from 'rxjs/operators';
import { of, interval } from 'rxjs';

//emit a value every 2 seconds
const source = interval(2000);
const example = source.pipe(
  //for demonstration, add 10 to and return as observable
  map(val => of(val + 10)),
  //merge values from inner observable
  concatAll()
);
//output: 'Example with Basic Observable 10', 'Example with Basic Observable 11'...
const subscribe = example.subscribe(val =>
  console.log('Example with Basic Observable:', val)
);
```

Example 2: concatAll with promise

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { map, concatAll } from 'rxjs/operators';
import { interval } from 'rxjs';

//create and resolve basic promise
const samplePromise = val => new Promise(resolve => resolve(val))
;
//emit a value every 2 seconds
const source = interval(2000);

const example = source.pipe(
  map(val => samplePromise(val)),
  //merge values from resolved promise
  concatAll()
);
//output: 'Example with Promise 0', 'Example with Promise 1'...
const subscribe = example.subscribe(val =>
  console.log('Example with Promise:', val)
);
```

Example 3: Delay while inner observables complete

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { take, concatAll } from 'rxjs/operators';
import { interval, of } from 'rxjs/observable/interval';

const obs1 = interval(1000).pipe(take(5));
const obs2 = interval(500).pipe(take(2));
const obs3 = interval(2000).pipe(take(1));
//emit three observables
const source = of(obs1, obs2, obs3);
//subscribe to each inner observable in order when previous completes
const example = source.pipe(concatAll());
/*
  output: 0,1,2,3,4,0,1,0
  How it works...
  Subscribes to each inner observable and emit values, when complete subscribe to next
  obs1: 0,1,2,3,4 (complete)
  obs2: 0,1 (complete)
  obs3: 0 (complete)
*/

const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Progress Bar](#)

Additional Resources

- [concatAll](#)  - Official docs
- [Flatten a higher order observable with concatAll in RxJS](#)   - André Staltz

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/concatAll.ts>

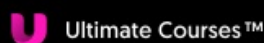
endWith

signature: `endWith(an: Values): Observable`

Emit given value(s) on completion.

💡 If you want to start with a value instead, check out `startWith` !

💡 If you want to perform an action on completion, but do not want to emit a value, check out `finalize` !



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Basic `endWith` **example**

([StackBlitz](#))

```
// RxJS v6+
import { endWith } from 'rxjs/operators';
import { of } from 'rxjs';

const source$ = of('Hello', 'Friend', 'Goodbye');

source$
  // emit on completion
  .pipe(endWith('Friend'))
  // 'Hello', 'Friend', 'Goodbye', 'Friend'
  .subscribe(console.log(val));
```

Example 2: endWith multiple values

([StackBlitz](#))

```
// RxJS v6+
import { endWith } from 'rxjs/operators';
import { of } from 'rxjs';

const source$ = of('Hello', 'Friend');

source$
  // emit on completion
  .pipe(endWith('Goodbye', 'Friend'))
  // 'Hello', 'Friend', 'Goodbye', 'Friend'
  .subscribe(console.log(val));
```

Example 3: Comparison to `finalize`

([StackBlitz](#))

```
// RxJS v6+
import { endWith, finalize } from 'rxjs/operators';
import { of } from 'rxjs';

const source$ = of('Hello', 'Friend');

source$
  // emit on completion
  .pipe(
    endWith('Goodbye', 'Friend'),
    // this function is invoked when unsubscribe methods are called
    finalize(() => console.log('Finally'))
  )
  // 'Hello', 'Friend', 'Goodbye', 'Friend'
  .subscribe(val => console.log(val));
// 'Finally'
```

Additional Resources

- [endWith](#)  - Official docs

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/endWith.ts>

forkJoin

signature: `forkJoin(...args, selector : function): Observable`

When all observables complete, emit the last emitted value from each.

💡 If you want corresponding emissions from multiple observables as they occur, try [zip](#)!

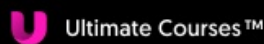
⚠️ If an inner observable does not complete `forkJoin` will never emit a value!

Why use `forkJoin` ?

This operator is best used when you have a group of observables and only care about the final emitted value of each. One common use case for this is if you wish to issue multiple requests on page load (or some other event) and only want to take action when a response has been received for all. In this way it is similar to how you might use `Promise.all`.

Be aware that if any of the inner observables supplied to `forkJoin` error you will lose the value of any other observables that would or have already completed if you do not `catch` the [error correctly on the inner observable](#). If you are only concerned with all inner observables completing successfully you can [catch the error on the outside](#).

It's also worth noting that if you have an observable that emits more than one item, and you are concerned with the previous emissions `forkJoin` is not the correct choice. In these cases you may be better off with an operator like [combineLatest](#) or [zip](#).



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: (v6.5+) Using a dictionary of sources

([StackBlitz](#))

```
// RxJS v6.5+
import { ajax } from 'rxjs/ajax';
import { forkJoin } from 'rxjs';

/*
  when all observables complete, provide the last
  emitted value from each as dictionary
*/
forkJoin(
  // as of RxJS 6.5+ we can use a dictionary of sources
  {
    google: ajax.getJSON('https://api.github.com/users/google'),
    microsoft: ajax.getJSON('https://api.github.com/users/microsoft'),
    users: ajax.getJSON('https://api.github.com/users')
  }
)
// { google: object, microsoft: object, users: array }
.subscribe(console.log);
```

Example 2: Observables completing after different durations

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { delay, take } from 'rxjs/operators';
import { forkJoin, of, interval } from 'rxjs';

const myPromise = val =>
  new Promise(resolve =>
    setTimeout(() => resolve(`Promise Resolved: ${val}`), 5000)
  );

/*
  when all observables complete, give the last
  emitted value from each as an array
*/
const example = forkJoin(
  //emit 'Hello' immediately
  of('Hello'),
  //emit 'World' after 1 second
  of('World').pipe(delay(1000)),
  //emit 0 after 1 second
  interval(1000).pipe(take(1)),
  //emit 0...1 in 1 second interval
  interval(1000).pipe(take(2)),
  //promise that resolves to 'Promise Resolved' after 5 seconds
  myPromise('RESULT')
);
//output: ["Hello", "World", 0, 1, "Promise Resolved: RESULT"]
const subscribe = example.subscribe(val => console.log(val));
```

Example 3: Making a variable number of requests

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { mergeMap } from 'rxjs/operators';
import { forkJoin, of } from 'rxjs';

const myPromise = val =>
  new Promise(resolve =>
    setTimeout(() => resolve(`Promise Resolved: ${val}`), 5000)
  );

const source = of([1, 2, 3, 4, 5]);
//emit array of all 5 results
const example = source.pipe(mergeMap(q => forkJoin(...q.map(myPr
omise))));
/*
  output:
  [
    "Promise Resolved: 1",
    "Promise Resolved: 2",
    "Promise Resolved: 3",
    "Promise Resolved: 4",
    "Promise Resolved: 5"
  ]
*/
const subscribe = example.subscribe(val => console.log(val));
```

Example 4: Handling errors on outside

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { delay, catchError } from 'rxjs/operators';
import { forkJoin, of, throwError } from 'rxjs';

/*
  when all observables complete, give the last
  emitted value from each as an array
*/
const example = forkJoin(
  //emit 'Hello' immediately
  of('Hello'),
  //emit 'World' after 1 second
  of('World').pipe(delay(1000)),
  // throw error
  _throw('This will error')
).pipe(catchError(error => of(error)));
//output: 'This will Error'
const subscribe = example.subscribe(val => console.log(val));
```

Example 5: Getting successful results when one inner observable errors

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { delay, catchError } from 'rxjs/operators';
import { forkJoin, of, throwError } from 'rxjs';

/*
  when all observables complete, give the last
  emitted value from each as an array
*/
const example = forkJoin(
  //emit 'Hello' immediately
  of('Hello'),
  //emit 'World' after 1 second
  of('World').pipe(delay(1000)),
  // throw error
  _throw('This will error').pipe(catchError(error => of(error)))
);
//output: ["Hello", "World", "This will error"]
const subscribe = example.subscribe(val => console.log(val));
```

Example 6: forkJoin in Angular

([plunker](#))

```
@Injectable()
export class MyService {
  makeRequest(value: string, delayDuration: number) {
    // simulate http request
    return of(`Complete: ${value}`).pipe(
      delay(delayDuration)
    );
  }
}

@Component({
  selector: 'my-app',
  template: `
    <div>
      <h2>forkJoin Example</h2>
      <ul>
        <li> {{propOne}} </li>
```

```
        <li> {{propTwo}} </li>
        <li> {{propThree}} </li>
    </ul>
</div>
    ,
})
export class App {
    public propOne: string;
    public propTwo: string;
    public propThree: string;
    constructor(private _myService: MyService) {}

    ngOnInit() {
        // simulate 3 requests with different delays
        forkJoin(
            this._myService.makeRequest('Request One', 2000),
            this._myService.makeRequest('Request Two', 1000),
            this._myService.makeRequest('Request Three', 3000)
        )
        .subscribe(([res1, res2, res3]) => {
            this.propOne = res1;
            this.propTwo = res2;
            this.propThree = res3;
        });
    }
}
```

Additional Resources

- [forkJoin](#)  - Official docs

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/observable/ForkJoinObservable.ts>

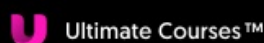
merge

signature: `merge(input: Observable): Observable`

Turn multiple observables into a single observable.

💡 This operator can be used as either a static or instance method!

💡 If order not throughput is a primary concern, try [concat](#) instead!



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: merging multiple observables, static method

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { mapTo } from 'rxjs/operators';
import { interval, merge } from 'rxjs';

//emit every 2.5 seconds
const first = interval(2500);
//emit every 2 seconds
const second = interval(2000);
//emit every 1.5 seconds
const third = interval(1500);
//emit every 1 second
const fourth = interval(1000);

//emit outputs from one observable
const example = merge(
  first.pipe(mapTo('FIRST!')),
  second.pipe(mapTo('SECOND!')),
  third.pipe(mapTo('THIRD')),
  fourth.pipe(mapTo('FOURTH'))
);
//output: "FOURTH", "THIRD", "SECOND!", "FOURTH", "FIRST!", "THI
RD", "FOURTH"
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: merge 2 observables, instance method

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { merge } from 'rxjs/operators';
import { interval } from 'rxjs';

//emit every 2.5 seconds
const first = interval(2500);
//emit every 1 second
const second = interval(1000);
//used as instance method
const example = first.pipe(merge(second));
//output: 0,1,0,2,...
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Battleship Game](#)
- [Flappy Bird Game](#)
- [HTTP Polling](#)
- [Lockscreen](#)
- [Memory Game](#)
- [Save Indicator](#)

Additional Resources

- [merge](#) 📄 - Official docs
- [Handling multiple streams with merge](#) 🖨️ 📄 - John Linquist
- [Sharing network requests with merge](#) 🖨️ 📄 - André Staltz
- [Combination operator: merge](#) 🖨️ 📄 - André Staltz
- [Build your own merge operator](#) 🖨️ - Kwinten Pisman



Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/merge.ts>

mergeAll

signature: `mergeAll(concurrent: number): Observable`

Collect and subscribe to all observables.


💡 In many cases you can use [mergeMap](#) as a single operator instead!

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

([example tests](#))

Example 1: mergeAll with promises

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { map, mergeAll } from 'rxjs/operators';
import { of } from 'rxjs';

const myPromise = val =>
  new Promise(resolve => setTimeout(() => resolve(`Result: ${val}`), 2000));
//emit 1,2,3
const source = of(1, 2, 3);

const example = source.pipe(
  //map each value to promise
  map(val => myPromise(val)),
  //emit result from source
  mergeAll()
);

/*
  output:
  "Result: 1"
  "Result: 2"
  "Result: 3"
*/
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: mergeAll with *concurrent* parameter

([StackBlitz](#) | [jsFiddle](#))




```
// RxJS v6+
import { take, map, delay, mergeAll } from 'rxjs/operators';
import { interval } from 'rxjs';

const source = interval(500).pipe(take(5));

/*
  interval is emitting a value every 0.5s. This value is then being mapped to interval that
  is delayed for 1.0s. The mergeAll operator takes an optional argument that determines how
  many inner observables to subscribe to at a time. The rest of the observables are stored
  in a backlog waiting to be subscribe.
*/
const example = source
  .pipe(
    map(val =>
      source.pipe(
        delay(1000),
        take(3)
      )
    ),
    mergeAll(2)
  )
  .subscribe(val => console.log(val));

/*
  The subscription is completed once the operator emits all values.
*/
```

Additional Resources

- [mergeAll](#)  - Official docs
- [Flatten a higher order observable with mergeAll in RxJS](#)   - André Staltz




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/mergeAll.ts>

pairwise

signature: `pairwise(): Observable<Array>`


Emit the previous and current values as an array.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1:

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { pairwise, take } from 'rxjs/operators';
import { interval } from 'rxjs';

//Returns: [0,1], [1,2], [2,3], [3,4], [4,5]
interval(1000)
  .pipe(
    pairwise(),
    take(5)
  )
  .subscribe(console.log);
```

Additional Resources

- [pairwise](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/pairwise.ts>

race

signature: `race(): Observable`

The observable to emit first is used.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: race with 4 observables

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { mapTo } from 'rxjs/operators';
import { interval } from 'rxjs/observable/interval';
import { race } from 'rxjs/observable/race';

//take the first observable to emit
const example = race(
  //emit every 1.5s
  interval(1500),
  //emit every 1s
  interval(1000).pipe(mapTo('1s won!')),
  //emit every 2s
  interval(2000),
  //emit every 2.5s
  interval(2500)
);
//output: "1s won!"..."1s won!"...etc
const subscribe = example.subscribe(val => console.log(val));
```


Example 2: race with an error

([StackBlitz](#) | [jsFiddle](#))

```
// RxJS v6+
import { delay, map } from 'rxjs/operators';
import { of, race } from 'rxjs';

//Throws an error and ignores the other observables.
const first = of('first').pipe(
  delay(100),
  map(_ => {
    throw 'error';
  })
);
const second = of('second').pipe(delay(200));
const third = of('third').pipe(delay(300));
// nothing logged
race(first, second, third).subscribe(val => console.log(val));
```

Additional Resources

- [race](#)  - Official docs



Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/race.ts>

startWith

signature: `startWith(an: Values): Observable`

Emit given value first.


💡 A [BehaviorSubject](#) can also start with an initial value!

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

([example tests](#))

Example 1: startWith on number sequence

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { startWith } from 'rxjs/operators';
import { of } from 'rxjs';

//emit (1,2,3)
const source = of(1, 2, 3);
//start with 0
const example = source.pipe(startWith(0));
//output: 0,1,2,3
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: startWith for initial scan value

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { startWith, scan } from 'rxjs/operators';
import { of } from 'rxjs';

//emit ('World!', 'Goodbye', 'World!')
const source = of('World!', 'Goodbye', 'World!');
//start with 'Hello', concat current string to previous
const example = source.pipe(
  startWith('Hello'),
  scan((acc, curr) => `${acc} ${curr}`)
);
/*
output:
"Hello"
"Hello World!"
"Hello World! Goodbye"
"Hello World! Goodbye World!"
*/
const subscribe = example.subscribe(val => console.log(val));
```

Example 3: startWith multiple values

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { startWith } from 'rxjs/operators';
import { interval } from 'rxjs';

//emit values in sequence every 1s
const source = interval(1000);
//start with -3, -2, -1
const example = source.pipe(startWith(-3, -2, -1));
//output: -3, -2, -1, 0, 1, 2....
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Platform Jumper Game](#)
- [Smart Counter](#)
- [Space Invaders Game](#)
- [Stop Watch](#)
- [Tank Battle Game](#)
- [Tetris Game](#)

Additional Resources

- [startWith](#) 📖 - Official docs
- [Displaying initial data with startWith](#) 🖥️💰 - John Linquist
- [Clear data while loading with startWith](#) 🖥️💰 - André Staltz
- [Combination operator: concat, startWith](#) 🖥️💰 - André Staltz

 Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/startWith.ts>

withLatestFrom

signature: `withLatestFrom(other: Observable, project: Function): Observable`

Also provide the last value from another observable.


💡 If you want the last emission any time a variable number of observables emits, try [combineLatest!](#)

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Latest value from quicker second source

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { withLatestFrom, map } from 'rxjs/operators';
import { interval } from 'rxjs';

//emit every 5s
const source = interval(5000);
//emit every 1s
const secondSource = interval(1000);
const example = source.pipe(
  withLatestFrom(secondSource),
  map(([first, second]) => {
    return `First Source (5s): ${first} Second Source (1s): ${second}`;
  })
);
/*
  "First Source (5s): 0 Second Source (1s): 4"
  "First Source (5s): 1 Second Source (1s): 9"
  "First Source (5s): 2 Second Source (1s): 14"
  ...
*/
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Slower second source

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { withLatestFrom, map } from 'rxjs/operators';
import { interval } from 'rxjs';

//emit every 5s
const source = interval(5000);
//emit every 1s
const secondSource = interval(1000);
//withLatestFrom slower than source
const example = secondSource.pipe(
  //both sources must emit at least 1 value (5s) before emitting
  withLatestFrom(source),
  map(([first, second]) => {
    return `Source (1s): ${first} Latest From (5s): ${second}`;
  })
);
/*
  "Source (1s): 4 Latest From (5s): 0"
  "Source (1s): 5 Latest From (5s): 0"
  "Source (1s): 6 Latest From (5s): 0"
  ...
*/
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Progress Bar](#)
- [Game Loop](#)
- [Space Invaders Game](#)

Additional Resources

- [withLatestFrom](#) 📖 - Official docs
- [Combination operator: withLatestFrom](#) 📺 📄 - André Staltz



Source Code:

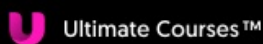
<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/withLatestFrom.ts>

zip

signature: `zip(observables: *): Observable`

After all observables emit, emit values as an array

💡 Combined with [interval](#) or [timer](#), zip can be used to time output from another source!



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: zip multiple observables emitting at alternate intervals

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { delay } from 'rxjs/operators';
import { of, zip } from 'rxjs';

const sourceOne = of('Hello');
const sourceTwo = of('World!');
const sourceThree = of('Goodbye');
const sourceFour = of('World!');
//wait until all observables have emitted a value then emit all
as an array
const example = zip(
  sourceOne,
  sourceTwo.pipe(delay(1000)),
  sourceThree.pipe(delay(2000)),
  sourceFour.pipe(delay(3000))
);
//output: ["Hello", "World!", "Goodbye", "World!"]
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: zip when 1 observable completes

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { take } from 'rxjs/operators';
import { interval, zip } from 'rxjs';

//emit every 1s
const source = interval(1000);
//when one observable completes no more values will be emitted
const example = zip(source, source.pipe(take(2)));
//output: [0,0]...[1,1]
const subscribe = example.subscribe(val => console.log(val));
```

Example 3: get X/Y coordinates of drag start/finish (mouse down/up)

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent, zip } from 'rxjs';
import { map } from 'rxjs/operators';

const documentEvent = eventName =>
  fromEvent(document, eventName).pipe(
    map((e: MouseEvent) => ({ x: e.clientX, y: e.clientY })))
  );

zip(documentEvent('mousedown'), documentEvent('mouseup')).subscribe(e =>
  console.log(JSON.stringify(e))
);
```

Example 4: mouse click duration

([StackBlitz](#))



```
// RxJS v6+
import { fromEvent, zip } from 'rxjs';
import { map } from 'rxjs/operators';

const eventTime = eventName =>
  fromEvent(document, eventName).pipe(map(() => new Date()));

const mouseClickDuration = zip(
  eventTime('mousedown'),
  eventTime('mouseup')
).pipe(map(([start, end]) => Math.abs(start.getTime() - end.getTime())));

mouseClickDuration.subscribe(console.log);
```

Additional Resources

- [zip](#)  - Official docs
- [Combination operator: zip](#)   - André Staltz



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/zip.ts>

Conditional Operators

For use-cases that depend on a specific condition to be met, these operators do the trick.


Contents

- [defaultIfEmpty](#)
- [every](#)
- [iif](#)
- [sequenceequal](#)

defaultIfEmpty

signature: `defaultIfEmpty(defaultValue: any): Observable`


Emit given value if nothing is emitted before completion.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Default for empty value

([Stackblitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { defaultIfEmpty } from 'rxjs/operators';
import { of } from 'rxjs';

//emit 'Observable.of() Empty!' when empty, else any values from
//source
const exampleOne = of().pipe(defaultIfEmpty('Observable.of() Empty!'));
//output: 'Observable.of() Empty!'
const subscribe = exampleOne.subscribe(val => console.log(val));
```

Example 2: Default for Observable.empty

([Stackblitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { defaultIfEmpty } from 'rxjs/operators';
import { empty } from 'rxjs';

//emit 'Observable.empty()!' when empty, else any values from source
const example = empty().pipe(defaultIfEmpty('Observable.empty()!'));
//output: 'Observable.empty()!'
const subscribe = example.subscribe(val => console.log(val));
```

Additional Resources

- [defaultIfEmpty](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/defaultIfEmpty.ts>

every

signature: `every(predicate: function, thisArg: any): Observable`


If all values pass predicate before completion emit true, else false.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone

Creator of LearnRxJS.io

Examples

Example 1: Some values false

([Stackblitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { every } from 'rxjs/operators';
import { of } from 'rxjs';

//emit 5 values
const source = of(1, 2, 3, 4, 5);
const example = source.pipe(
  //is every value even?
  every(val => val % 2 === 0)
);
//output: false
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: All values true

([Stackblitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { every } from 'rxjs/operators';
import { of } from 'rxjs';

//emit 5 values
const allEvens = of(2, 4, 6, 8, 10);
const example = allEvens.pipe(
  //is every value even?
  every(val => val % 2 === 0)
);
//output: true
const subscribe = example.subscribe(val => console.log(val));
```

Example 3: Values arriving over time and completing stream prematurely due to every returning false

([Stackblitz](#))



```
// RxJS v6+
console.clear();
import { concat, of } from 'rxjs';
import { every, delay, tap } from 'rxjs/operators';

const log = console.log;
const returnCode = request => (Number.isInteger(request) ? 200 : 400);
const fakeRequest = request =>
  of({ code: returnCode(request) }).pipe(
    tap(_ => log(request)),
    delay(1000)
  );

const apiCalls$ = concat(
  fakeRequest(1),
  fakeRequest('invalid payload'),
  fakeRequest(2) //this won't execute as every will return false
  for previous line
).pipe(
  every(e => e.code === 200),
  tap(e => log(`all request successful: ${e}`))
);

apiCalls$.subscribe();
```

Additional Resources

- [every](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/every.ts>

iif

signature: `iif(condition: () => boolean, trueResult: SubscribableOrPromise = EMPTY, falseResult: SubscribableOrPromise = EMPTY): Observable`


Subscribe to first or second observable based on a condition

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: simple iif

([Stackblitz](#))

```
// RxJS v6+
import { iif, of, interval } from 'rxjs';
import { mergeMap } from 'rxjs/operators';

const r$ = of('R');
const x$ = of('X');

interval(1000)
  .pipe(mergeMap(v => iif(() => v % 4 === 0, r$, x$)))
  .subscribe(console.log);

// output: R, X, X, X, R, X, X, X, etc...
```

Example 2: iif with mouse moves

([Stackblitz](#))

```
// RxJS v6+
import { fromEvent, iif, of } from 'rxjs';
import { mergeMap, map, throttleTime, filter } from 'rxjs/operators';

const r$ = of(`I'm saying R!!`);
const x$ = of(`X's always win!!`);

fromEvent(document, 'mousemove')
  .pipe(
    throttleTime(50),
    filter((move: MouseEvent) => move.clientY < 210),
    map((move: MouseEvent) => move.clientY),
    mergeMap(yCoord => iif(() => yCoord < 110, r$, x$))
  )
  .subscribe(console.log);
```

Example 3: iif with default

([Stackblitz](#))

```
// RxJS v6+
import { fromEvent, iif, of, interval, pipe } from 'rxjs';
import { mergeMap } from 'rxjs/operators';

interval(1000)
  .pipe(
    mergeMap(v =>
      iif(
        () => !(v % 2),
        of(v)
        // if not supplied defaults to EMPTY
      )
    )
  )
  // output: 1,3,5...
  .subscribe(console.log);
```

Related Recipes

- [Swipe To Refresh](#)

Additional Resources

- [iif](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/iif.ts>

sequenceEqual

signature: `sequenceEqual(compareTo: Observable, comparator?: (a, b) => boolean): Observable`


Compares emitted sequence to expected sequence for match

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: simple sequenceEqual

([Stackblitz](#))

```
// RxJS v6+
import { of, from } from 'rxjs';
import { sequenceEqual, switchMap } from 'rxjs/operators';

const expectedSequence = from([4, 5, 6]);

of([1, 2, 3], [4, 5, 6], [7, 8, 9])
  .pipe(switchMap(arr => from(arr).pipe(sequenceEqual(expectedSequence))))
  .subscribe(console.log);

//output: false, true, false
```

Example 2: sequenceEqual with keyboard events

([Stackblitz](#))

```
// RxJS v6+
import { from, fromEvent } from 'rxjs';
import { sequenceEqual, map, bufferCount, mergeMap, tap } from 'rxjs/operators';

const expectedSequence = from(['q', 'w', 'e', 'r', 't', 'y']);
const setResult = text => (document.getElementById('result').innerHTMLText = text);

fromEvent(document, 'keydown')
  .pipe(
    map((e: KeyboardEvent) => e.key),
    tap(v => setResult(v)),
    bufferCount(6),
    mergeMap(keyDowns =>
      from(keyDowns).pipe(
        sequenceEqual(expectedSequence),
        tap(isItQwerty => setResult(isItQwerty ? 'WELL DONE!' : 'TYPE AGAIN!'))
      )
    )
  )
  .subscribe(e => console.log(`did you say qwerty? ${e}`));
```

Related Recipes

- [Lockscreen](#)
- [Memory Game](#)

Additional Resources

- [sequenceEqual](#)  - Official docs



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/sequenceEqual.ts>

Creation Operators

These operators allow the creation of an observable from nearly anything. From generic to specific use-cases you are free, and encouraged, to turn [everything into a stream](#).

Contents

- [ajax](#) ★
- [create](#)
- [defer](#)
- [empty](#)
- [from](#) ★
- [fromEvent](#)
- [generate](#)
- [interval](#)
- [of](#) ★
- [range](#)
- [throw](#)
- [timer](#)

★ - *commonly used*


Additional Resources

- [Creating Observables From Scratch](#) 📄 📖 - André Staltz

ajax

signature: `ajax(urlOrRequest: string | AjaxRequest)`


Create an observable for an Ajax request with either a request object with url, headers, etc or a string for a URL.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Observable that emits the response object that is being returned from the request.

([StackBlitz](#))

```
// RxJS v6+
import { ajax } from 'rxjs/ajax';

const githubUsers = `https://api.github.com/users?per_page=2`;

const users = ajax(githubUsers);

const subscribe = users.subscribe(
  res => console.log(res),
  err => console.error(err)
```

```
);

/* output
{
  "originalEvent":{
    "isTrusted":true
  },
  "xhr":{

  },
  "request":{
    "async":true,
    "crossDomain":true,
    "withCredentials":false,
    "headers":{

    },
    "method":"GET",
    "responseType":"json",
    "timeout":0,
    "url":"https://api.github.com/users?per_page=2"
  },
  "status":200,
  "responseType":"json",
  "response":[
    {
      "login":"mojombo",
      "id":1,
      "node_id":"MDQ6VXNlcjE=",
      "avatar_url":"https://avatars0.githubusercontent.com/u/1?v
=4",
      "gravatar_id": "",
      ...
    },
    {
      "login":"defunkt",
      "id":2,
      "node_id":"MDQ6VXNlcjI=",
      "avatar_url":"https://avatars0.githubusercontent.com/u/2?v
=4",
      "gravatar_id": "",
      ...
    }
  ]
}
```

```
}  
]  
}  
*/
```

Example 2: Observable that emits only the json key of the response object that is being returned from the request.

([StackBlitz](#))

```
// RxJS v6+
import { ajax } from 'rxjs/ajax';

const githubUsers = `https://api.github.com/users?per_page=2`;

const users = ajax.getJSON(githubUsers);

const subscribe = users.subscribe(
  res => console.log(res),
  err => console.error(err)
);

/* output
[
  {
    "login": "mojombo",
    "id": 1,
    "node_id": "MDQ6VXNlcjE=",
    "avatar_url": "https://avatars0.githubusercontent.com/u/1?v=4",
    "gravatar_id": "",
    "...",
  },
  {
    "login": "defunkt",
    "id": 2,
    "node_id": "MDQ6VXNlcjI=",
    "avatar_url": "https://avatars0.githubusercontent.com/u/2?v=4",
    "gravatar_id": "",
    "...",
  }
]
*/
```

Example 3: Observable that emits the error object that is being returned from the request.

([StackBlitz](#))

```
// RxJS v6+
import { ajax } from 'rxjs/ajax';

const githubUsers = `https://api.github.com/error`;

const users = ajax.getJSON(githubUsers);

const subscribe = users.subscribe(
  res => console.log(res),
  err => console.error(err)
);

/* output
Error: ajax error 404
*/
```

Example 4: Ajax operator with object as input.

([StackBlitz](#))


```
// RxJS v6+
import { ajax } from 'rxjs/ajax';

const githubUsers = `https://api.github.com/error`;

const users = ajax({
  url: githubUsers,
  method: 'GET',
  headers: {
    /*some headers*/
  },
  body: {
    /*in case you need a body*/
  }
});

const subscribe = users.subscribe(
  res => console.log(res),
  err => console.error(err)
);
```

Additional Resources

- [ajax](#)  - Official docs



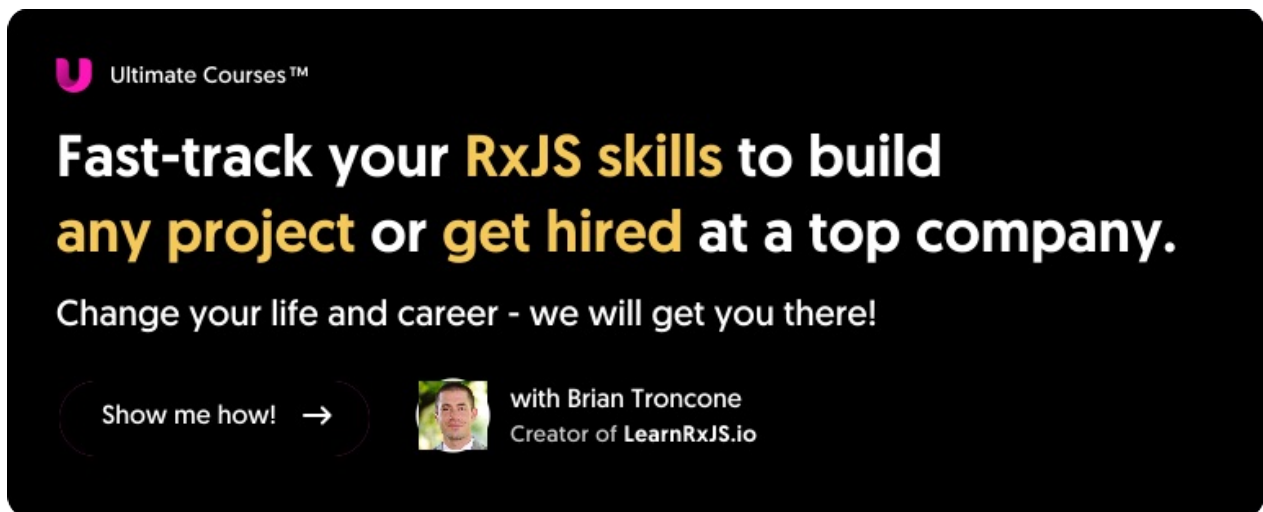
Source Code:

<https://github.com/ReactiveX/rxjs/blob/6.4.0/src/internal/observable/dom/ajax.ts#L20-L19>

create

signature: `create(subscribe: function)`

Create an observable with given subscription function.




U Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Observable that emits multiple values

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { Observable } from 'rxjs';
/*
  Create an observable that emits 'Hello' and 'World' on
  subscription.
*/
const hello = Observable.create(function(observer) {
  observer.next('Hello');
  observer.next('World');
  observer.complete();
});

//output: 'Hello'...'World'
const subscribe = hello.subscribe(val => console.log(val));
```

Example 2: Observable that emits even numbers on timer

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { Observable } from 'rxjs';

/*
  Increment value every 1s, emit even numbers.
*/
const evenNumbers = Observable.create(function(observer) {
  let value = 0;
  const interval = setInterval(() => {
    if (value % 2 === 0) {
      observer.next(value);
    }
    value++;
  }, 1000);

  return () => clearInterval(interval);
});
//output: 0...2...4...6...8
const subscribe = evenNumbers.subscribe(val => console.log(val));

//unsubscribe after 10 seconds
setTimeout(() => {
  subscribe.unsubscribe();
}, 10000);
```

Additional Resources

- [create](#) 📖 - Official docs
- [Creation operators: Create\(\)](#) 🖥️ 📄 - André Staltz
- [Using Observable.create for fine-grained control](#) 🖥️ 📄 - Shane Osbourne



Source Code:

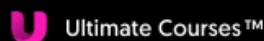
<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/GenerateObservable.ts>

defer

signature: `defer(observableFactory: function():
SubscribableOrPromise): Observable`

Create an observable with given subscription function.

💡 `defer` is used as part of the `iif` operator!



Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: Defer to get current date/time at the time of subscription

([StackBlitz](#))

```
// RxJS v6+
import { defer, of, timer, merge } from 'rxjs';
import { switchMap } from 'rxjs/operators';

const s1 = of(new Date()); //will capture current date time
const s2 = defer(() => of(new Date())); //will capture date time
    at the moment of subscription

console.log(new Date());

timer(2000)
    .pipe(switchMap(_ => merge(s1, s2)))
    .subscribe(console.log);


/*
OUTPUT =>
2019-02-10T12:38:30.000Z (current date/time from first console
log)
2019-02-10T12:38:30.000Z (date/time in s1 console log, captured
date/time at the moment of observable creation)
2019-02-10T12:38:32.000Z (date/time in s2 console log, captured
date/time at the moment of subscription)
*/

/*//NOTE: 'traditional' js equivalent of timer code above is:
setTimeout(() => {
    s1.subscribe(console.log);
    s2.subscribe(console.log);
}, 2000);
*/
```

Related Recipes

- [Save Indicator](#)

Additional Resources

- [defer](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/defer.ts>

empty

signature: `empty(scheduler: Scheduler): Observable`


Observable that immediately completes.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: empty immediately completes

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { empty } from 'rxjs';

//output: 'Complete!'
const subscribe = empty().subscribe({
  next: () => console.log('Next'),
  complete: () => console.log('Complete!')
});
```

Example 2: empty with timer

([StackBlitz](#))

```
// RxJS v6+
import { interval, fromEvent, merge, empty } from 'rxjs';
import { switchMap, scan, takeWhile, startWith, mapTo } from 'rxjs/operators';

const countdownSeconds = 10;
const setHTML = id => val => (document.getElementById(id).innerHTML = val);
const pauseButton = document.getElementById('pause');
const resumeButton = document.getElementById('resume');
const interval$ = interval(1000).pipe(mapTo(-1));




const pause$ = fromEvent(pauseButton, 'click').pipe(mapTo(false));
const resume$ = fromEvent(resumeButton, 'click').pipe(mapTo(true));

const timer$ = merge(pause$, resume$)
  .pipe(
    startWith(true),
    // if timer is paused return empty observable
    switchMap(val => (val ? interval$ : empty())),
    scan((acc, curr) => (curr ? curr + acc : acc), countdownSeconds),
    takeWhile(v => v >= 0)
  )
  .subscribe(setHTML('remaining'));
```

Related Recipes

- [Memory Game](#)
- [Save Indicator](#)

Additional Resources

- [empty](#)  - Official docs
- [Creation operators: empty, never, and throw](#)   - André Staltz



Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/EmptyObservable.ts>

from

signature: `from(ish: ObservableInput, mapFn: function, thisArg: any, scheduler: Scheduler): Observable`

Turn an array, promise, or iterable into an observable.


- 💡 This operator can be used to convert a promise to an observable!
 - 💡 For arrays and iterables, all contained values will be emitted as a sequence!
 - 💡 This operator can also be used to emit a string as a sequence of characters!
-

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

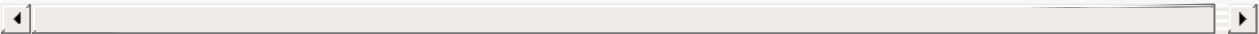
Examples

Example 1: Observable from array

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';

//emit array as a sequence of values
const arraySource = from([1, 2, 3, 4, 5]);
//output: 1,2,3,4,5
const subscribe = arraySource.subscribe(val => console.log(val));
```



Example 2: Observable from promise

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';

//emit result of promise
const promiseSource = from(new Promise(resolve => resolve('Hello World')));
//output: 'Hello World'
const subscribe = promiseSource.subscribe(val => console.log(val));
```



Example 3: Observable from collection

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';

//works on js collections
const map = new Map();
map.set(1, 'Hi');
map.set(2, 'Bye');

const mapSource = from(map);
//output: [1, 'Hi'], [2, 'Bye']
const subscribe = mapSource.subscribe(val => console.log(val));
```

Example 4: Observable from string

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';

//emit string as a sequence
const source = from('Hello World');
//output: 'H','e','l','l','o',' ',' ','W','o','r','l','d'
const subscribe = source.subscribe(val => console.log(val));
```

Related Recipes

- [HTTP Polling](#)
- [Lockscreen](#)
- [Memory Game](#)
- [Progress Bar](#)

Additional Resources

- [from](#)  - Official docs
- [Creation operators: from, fromArray, fromPromise](#)   - André Staltz


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/from.ts>

fromEvent

signature: `fromEvent(target: EventTargetLike, eventName: string, selector: function): Observable`


Turn event into observable sequence.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Observable from mouse clicks

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { map } from 'rxjs/operators';

//create observable that emits click events
const source = fromEvent(document, 'click');
//map to string with given event timestamp
const example = source.pipe(map(event => `Event time: ${event.timeStamp}`));
//output (example): 'Event time: 7276.3900000000001'
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Catch The Dot Game](#)
- [Click Ninja Game](#)
- [Flappy Bird Game](#)
- [Game Loop](#)
- [Horizontal Scroll Indicator](#)
- [HTTP Polling](#)
- [Lockscreen](#)
- [Memory Game](#)
- [Mine Sweeper Game](#)
- [Platform Jumper Game](#)
- [Progress Bar](#)
- [Save Indicator](#)
- [Smart Counter](#)
- [Space Invaders Game](#)
- [Stop Watch](#)
- [Swipe To Refresh](#)
- [Tank Battle Game](#)
- [Tetris Game](#)
- [Type Ahead](#)

Additional Resources

- [fromEvent](#)  - Official docs



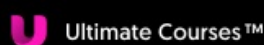
Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/fromEvent.ts>

generate

signature: `generate(initialStateOrOptions: GenerateOptions, condition?: ConditionFunc, iterate?: IterateFunc, resultSelectorOrObservable?: (ResultFunc) | SchedulerLike, scheduler?: SchedulerLike): Observable`

Generates an observable sequence by running a state-driven loop producing the sequence's elements, using the specified scheduler to send out observer messages.



Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Generate

([StackBlitz](#))

Related Recipes

- [Breakout Game](#)
- [Memory Game](#)

Additional Resources

- [generate](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/generate.ts>

interval

signature: `interval(period: number, scheduler: Scheduler): Observable`


Emit numbers in sequence based on provided timeframe.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Emit sequence of values at 1 second interval

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';

//emit value in sequence every 1 second
const source = interval(1000);
//output: 0,1,2,3,4,5....
const subscribe = source.subscribe(val => console.log(val));
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Catch The Dot Game](#)
- [Flappy Bird Game](#)
- [Matrix Digital Rain](#)
- [Memory Game](#)
- [Platform Jumper Game](#)
- [Space Invaders Game](#)
- [Stop Watch](#)
- [Tank Battle Game](#)
- [Tetris Game](#)

Additional Resources

- [interval](#) 📄 - Official docs
- [Creation operators: interval and timer](#) 🖥️ 💰 - André Staltz
- [Build your own interval operator](#) 🖥️ - Kwinten Pisman



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/interval.ts>

of / just

signature: `of(...values, scheduler: Scheduler): Observable`

Emit variable amount of values in a sequence and then emits a complete notification.

Examples

Example 1: Emitting a sequence of numbers

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
//emits any number of provided values in sequence
const source = of(1, 2, 3, 4, 5);
//output: 1,2,3,4,5
const subscribe = source.subscribe(val => console.log(val));
```

Example 2: Emitting an object, array, and function

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))





```
// RxJS v6+
import { of } from 'rxjs';
//emits values of any type
const source = of({ name: 'Brian' }, [1, 2, 3], function hello() {
  return 'Hello';
});
//output: {name: 'Brian'}, [1,2,3], function hello() { return 'Hello' }
const subscribe = source.subscribe(val => console.log(val));
```



Related Recipes

- [Battleship Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Mine Sweeper Game](#)
- [Platform Jumper Game](#)
- [Save Indicator](#)
- [Swipe To Refresh](#)
- [Tetris Game](#)
- [Type Ahead](#)

Additional Resources

- [of](#)  - Official docs
- [Creation operators: of](#)   - André Staltz
- [Build your own of operator](#)  - Kwinten Pisman




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/of.ts>

range

signature: `range(start: number, count: number, scheduler: Scheduler): Observable`


Emit numbers in provided range in sequence.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples


Example 1: Emit range 1-10

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { range } from 'rxjs';

//emit 1-10 in sequence
const source = range(1, 10);
//output: 1,2,3,4,5,6,7,8,9,10
const example = source.subscribe(val => console.log(val));
```

Additional Resources

- [range](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/range.ts>

throw

signature: `throw(error: any, scheduler: Scheduler): Observable`


Emit error on subscription.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Throw error on subscription

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { throwError } from 'rxjs';

//emits an error with specified value on subscription
const source = throwError('This is an error!');
//output: 'Error: This is an error!'
const subscribe = source.subscribe({
  next: val => console.log(val),
  complete: () => console.log('Complete!'),
  error: val => console.log(`Error: ${val}`)
});
```


Related Examples

- [Throwing after 3 retries](#)

Additional Resources

- [throw](#) 📄 - Official docs
- [Creation operators: empty, never, and throw](#) 🖥️ 📄 - André Staltz



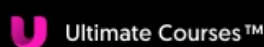
Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/throwError.ts>

timer

signature: `timer(initialDelay: number | Date, period: number, scheduler: Scheduler): Observable`

After given duration, emit numbers in sequence every specified duration.



Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: timer emits 1 value then completes

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { timer } from 'rxjs';

//emit 0 after 1 second then complete, since no second argument
//is supplied
const source = timer(1000);
//output: 0
const subscribe = source.subscribe(val => console.log(val));
```

Example 2: timer emits after 1 second, then every 2 seconds

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { timer } from 'rxjs';

/*
  timer takes a second argument, how often to emit subsequent va
  lues
  in this case we will emit first value after 1 second and subse
  quent
  values every 2 seconds after
*/
const source = timer(1000, 2000);
//output: 0,1,2,3,4,5.....
const subscribe = source.subscribe(val => console.log(val));
```

Related Recipes

- [HTTP Polling](#)

Additional Resources

- [timer](#) 📖 - Official docs
- [Creation operators: interval and timer](#) 📖 💡 - André Staltz
- [Build your own timer operator](#) 📖 - Kwinten Pisman

📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/timer.ts>

Error Handling Operators

Errors are an unfortunate side-effect of development. These operators provide effective ways to gracefully handle errors and retry logic, should they occur.

Contents

- [catch / catchError](#) ★
- [retry](#)
- [retryWhen](#)


★ - *commonly used*

catch / catchError

signature: `catchError(project : function): Observable`

Gracefully handle errors in an observable sequence.


⚠ Remember to return an observable from the catchError function!

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

([example tests](#))

Example 1: Catching error from observable

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { throwError, of } from 'rxjs';
import { catchError } from 'rxjs/operators';
//emit error
const source = throwError('This is an error!');
//gracefully handle error, returning observable with error message
const example = source.pipe(catchError(val => of(`I caught: ${val}`)));
//output: 'I caught: This is an error'
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Catching rejected promise

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { timer, from, of } from 'rxjs';
import { mergeMap, catchError } from 'rxjs/operators';

//create promise that immediately rejects
const myBadPromise = () =>
  new Promise((resolve, reject) => reject('Rejected!'));
//emit single value after 1 second
const source = timer(1000);
//catch rejected promise, returning observable containing error message
const example = source.pipe(
  mergeMap(_ =>
    from(myBadPromise()).pipe(catchError(error => of(`Bad Promise: ${error}`)))
  )
);
//output: 'Bad Promise: Rejected'
const subscribe = example.subscribe(val => console.log(val));
```

Example 3: Catching errors comparison when using switchMap/mergeMap/concatMap/exhaustMap

([StackBlitz](#))

```
// switchMap in example below can be replaced with mergeMap/conc
atMap/exhaustMap, the same behaviour applies
import { throwError, fromEvent, of } from 'rxjs';
import {
  catchError,
  tap,
  switchMap,
  mergeMap,
  concatMap,
  exhaustMap
} from 'rxjs/operators';




const fakeRequest$ = of().pipe(
  tap(_ => console.log('fakeRequest')),
  throwError
);

const iWillContinueListening$ = fromEvent(
  document.getElementById('continued'),
  'click'
).pipe(
  switchMap(_ => fakeRequest$.pipe(catchError(_ => of('keep on c
licking!!!'))))
);

const iWillStopListening$ = fromEvent(
  document.getElementById('stopped'),
  'click'
).pipe(
  switchMap(_ => fakeRequest$),
  catchError(_ => of('no more requests!!!'))
);

iWillContinueListening$.subscribe(console.log);
iWillStopListening$.subscribe(console.log);
```

Additional Resources

- [catchError](#)  - Official docs
 - [Error handling operator: catch](#)   - André Staltz
-

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/catchError.ts>

retry

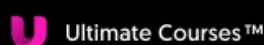
signature: `retry(number: number): Observable`

Retry an observable sequence a specific number of times should an error occur.

💡 Useful for retrying HTTP requests!

💡 If you only want to retry in certain cases, check out `retryWhen` !

💡 For non error cases check out `repeat` !



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Retry 2 times on error

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, of, throwError } from 'rxjs';
import { mergeMap, retry } from 'rxjs/operators';

//emit value every 1s
const source = interval(1000);
const example = source.pipe(
  mergeMap(val => {
    //throw error for demonstration
    if (val > 5) {
      return throwError('Error!');
    }
    return of(val);
  }),
  //retry 2 times on error
  retry(2)
);
/*
output:
0..1..2..3..4..5..
0..1..2..3..4..5..
0..1..2..3..4..5..
"Error!: Retried 2 times then quit!"
*/
const subscribe = example.subscribe({
  next: val => console.log(val),
  error: val => console.log(`${val}: Retried 2 times then quit!`)
});
```

Additional Resources

- [retry](#) 📖 - Official docs
- [Error handling operator: retry and retryWhen](#) 📖 💡 - André Staltz


📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/retry.ts>

retryWhen

signature: `retryWhen(receives: (errors: Observable) => Observable, the: scheduler): Observable`


Retry an observable sequence on error based on custom criteria.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone

Creator of LearnRxJS.io

Examples

Example 1: Trigger retry after specified duration

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { timer, interval } from 'rxjs';
import { map, tap, retryWhen, delayWhen } from 'rxjs/operators';

//emit value every 1s
const source = interval(1000);
const example = source.pipe(
  map(val => {
    if (val > 5) {
      //error will be picked up by retryWhen
      throw val;
    }
    return val;
  }),
  retryWhen(errors =>
    errors.pipe(
      //log error message
      tap(val => console.log(`Value ${val} was too high!`)),
      //restart in 6 seconds
      delayWhen(val => timer(val * 1000))
    )
  )
);
/*
output:
0
1
2
3
4
5
"Value 6 was too high!"
--Wait 6 seconds then repeat
*/
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Customizable retry with increased duration

([StackBlitz](#))

```
import { Observable, throwError, timer } from 'rxjs';
import { mergeMap, finalize } from 'rxjs/operators';

export const genericRetryStrategy = ({
  maxRetryAttempts = 3,
  scalingDuration = 1000,
  excludedStatusCodes = []
}: {
  maxRetryAttempts?: number,
  scalingDuration?: number,
  excludedStatusCodes?: number[]
} = {}) => (attempts: Observable<any>) => {
  return attempts.pipe(
    mergeMap((error, i) => {
      const retryAttempt = i + 1;
      // if maximum number of retries have been met
      // or response is a status code we don't wish to retry, th
      row error
      if (
        retryAttempt > maxRetryAttempts ||
        excludedStatusCodes.find(e => e === error.status)
      ) {
        return throwError(error);
      }
      console.log(
        `Attempt ${retryAttempt}: retrying in ${retryAttempt *
          scalingDuration}ms`
      );
      // retry after 1s, 2s, etc...
      return timer(retryAttempt * scalingDuration);
    }),
    finalize(() => console.log('We are done!'))
  );
};
```

```
import { Component, OnInit } from '@angular/core';
import { catchError, retryWhen } from 'rxjs/operators';
import { of } from 'rxjs';
import { genericRetryStrategy } from './rxjs-utils';
import { AppService } from './app.service';



@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: [ './app.component.css' ]
})
export class AppComponent implements OnInit {
  constructor(private _appService: AppService) {}

  ngOnInit() {
    this._appService
      .getData(500)
      .pipe(
        retryWhen(genericRetryStrategy()),
        catchError(error => of(error))
      )
      .subscribe(console.log);

    // excluding status code, delay for logging clarity
    setTimeout(() => {
      this._appService
        .getData(500)
        .pipe(
          retryWhen(genericRetryStrategy({
            scalingDuration: 2000,
            excludedStatusCodes: [500]
          })),
          catchError(error => of(error))
        )
        .subscribe(e => console.log('Exluded code:', e.status));

    }, 8000);
  }
}
```

Additional Resources

- [retryWhen](#)  - Official docs
 - [Error handling operator: retry and retryWhen](#)   - André Staltz
-

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/retryWhen.ts>

Multicasting Operators

In RxJS observables are cold, or unicast by default. These operators can make an observable hot, or multicast, allowing side-effects to be shared among multiple subscribers.

Contents

- [publish](#)
- [multicast](#)
- [share](#) ★
- [shareReplay](#) ★

★ - *commonly used*


Additional Resources

- [Hot vs Cold Observables](#) 📄 - Ben Lesh
- [Unicast v Multicast](#) 📄 - GitHub Discussion
- [Demystifying Hot and Cold Observables](#) 📺 - André Staltz

publish

signature: `publish() : ConnectableObservable`

Share source and make hot by calling connect.




U Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Connect observable after subscribers

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { publish, tap } from 'rxjs/operators';

//emit value every 1 second
const source = interval(1000);
const example = source.pipe(
  //side effects will be executed once
  tap(_ => console.log('Do Something!')),
  //do nothing until connect() is called
  publish()
);

/*
  source will not emit values until connect() is called
  output: (after 5s)
  "Do Something!"
  "Subscriber One: 0"
  "Subscriber Two: 0"
  "Do Something!"
  "Subscriber One: 1"
  "Subscriber Two: 1"
*/
const subscribe = example.subscribe(val =>
  console.log(`Subscriber One: ${val}`)
);
const subscribeTwo = example.subscribe(val =>
  console.log(`Subscriber Two: ${val}`)
);

//call connect after 5 seconds, causing source to begin emitting
  items
setTimeout(() => {
  example.connect();
}, 5000);
```

Additional Resources

- [publish](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/publish.ts>

multicast

signature: `multicast(selector: Function): Observable`

Share source utilizing the provided Subject.




U Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: multicast with standard Subject

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { Subject, interval } from 'rxjs';
import { take, tap, multicast, mapTo } from 'rxjs/operators';

//emit every 2 seconds, take 5
const source = interval(2000).pipe(take(5));

const example = source.pipe(
  //since we are multicasting below, side effects will be executed once
  tap(() => console.log('Side Effect #1')),
  mapTo('Result!')
);

//subscribe subject to source upon connect()
const multi = example.pipe(multicast(() => new Subject()));
/*
  subscribers will share source
  output:
  "Side Effect #1"
  "Result!"
  "Result!"
  ...
*/
const subscriberOne = multi.subscribe(val => console.log(val));
const subscriberTwo = multi.subscribe(val => console.log(val));
//subscribe subject to source
multi.connect();
```

Example 2: multicast with ReplaySubject

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, ReplaySubject } from 'rxjs';
import { take, multicast, tap, mapTo } from 'rxjs/operators';

//emit every 2 seconds, take 5
const source = interval(2000).pipe(take(5));

//example with ReplaySubject
const example = source.pipe(
  //since we are multicasting below, side effects will be executed once
  tap(_ => console.log('Side Effect #2')),
  mapTo('Result Two!')
);
//can use any type of subject
const multi = example.pipe(multicast(() => new ReplaySubject(5)))
;
//subscribe subject to source
multi.connect();

setTimeout(() => {
  /*
    subscriber will receive all previous values on subscription
    because
    of ReplaySubject
    */
  const subscriber = multi.subscribe(val => console.group(val));
}, 5000);
```

Additional Resources

- [multicast](#)  - Official docs

 Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/multicast.ts>

share

signature: `share(): Observable`

Share source among multiple subscribers.


💡 share is like [multicast](#) with a Subject and refCount!

 Ultimate Courses™

**Fast-track your [RxJS skills](#) to build
[any project](#) or [get hired](#) at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Multiple subscribers sharing source

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { timer } from 'rxjs';
import { tap, mapTo, share } from 'rxjs/operators';

//emit value in 1s
const source = timer(1000);
//log side effect, emit result
const example = source.pipe(
  tap(() => console.log('***SIDE EFFECT***')),
  mapTo('***RESULT***')
);




/*
  ***NOT SHARED, SIDE EFFECT WILL BE EXECUTED TWICE***
  output:
  ***SIDE EFFECT***
  ***RESULT***
  ***SIDE EFFECT***
  ***RESULT***
*/
const subscribe = example.subscribe(val => console.log(val));
const subscribeTwo = example.subscribe(val => console.log(val));

//share observable among subscribers
const sharedExample = example.pipe(share());
/*
  ***SHARED, SIDE EFFECT EXECUTED ONCE***
  output:
  ***SIDE EFFECT***
  ***RESULT***
  ***RESULT***
*/
const subscribeThree = sharedExample.subscribe(val => console.log(
  val));
const subscribeFour = sharedExample.subscribe(val => console.log(
  val));
```

Related Recipes

- [Progress Bar](#)
- [Game Loop](#)
- [Save Indicator](#)

Additional Resources

- [share](#)  - Official docs
- [Sharing streams with share](#)   - John Linquist

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/share.ts>

shareReplay

signature: `shareReplay(bufferSize?: number, windowTime?: number, scheduler?I IScheduler): Observable`

Share source and replay specified number of emissions on subscription.

Why use `shareReplay` ?

You generally want to use `shareReplay` when you have side-effects or taxing computations that you do not wish to be executed amongst multiple subscribers. It may also be valuable in situations where you know you will have late subscribers to a stream that need access to previously emitted values. This ability to *replay* values on subscription is what differentiates `share` and `shareReplay`.

For instance, suppose you have an observable that emits the last visited url. In the first example we are going to use `share` :

```
// simulate url change with subject
const routeEnd = new Subject<{data: any, url: string}>();

// grab url and share with subscribers
const lastUrl = routeEnd.pipe(
  pluck('url'),
  share()
);

// initial subscriber required
const initialSubscriber = lastUrl.subscribe(console.log);

// simulate route change
routeEnd.next({data: {}, url: 'my-path'});

// nothing logged
const lateSubscriber = lastUrl.subscribe(console.log);
```

In the above example nothing is logged as the `lateSubscriber` subscribes to the source. Now suppose instead we wanted to give access to the last emitted value on subscription, we can accomplish this with `shareReplay` :

```
import { Subject } from 'rxjs/Subject';
import { ReplaySubject } from 'rxjs/ReplaySubject';
import { pluck, share, shareReplay, tap } from 'rxjs/operators';

// simulate url change with subject
const routeEnd = new Subject<{data: any, url: string}>();

// grab url and share with subscribers
const lastUrl = routeEnd.pipe(
  tap(_ => console.log('executed')),
  pluck('url'),
  // defaults to all values so we set it to just keep and replay
  // last one
  shareReplay(1)
);

// requires initial subscription
const initialSubscriber = lastUrl.subscribe(console.log);

// simulate route change
// logged: 'executed', 'my-path'
routeEnd.next({data: {}, url: 'my-path'});

// logged: 'my-path'
const lateSubscriber = lastUrl.subscribe(console.log);
```

Note that this is similar behavior to what you would see if you subscribed a `ReplaySubject` to the `lastUrl` stream, then subscribed to that `Subject` :

```
// simulate url change with subject
const routeEnd = new Subject<{data: any, url: string}>();

// instead of using shareReplay, use ReplaySubject
const shareWithReplay = new ReplaySubject();

// grab url and share with subscribers
const lastUrl = routeEnd.pipe(
  pluck('url')
)
.subscribe(val => shareWithReplay.next(val));

// simulate route change
routeEnd.next({data: {}, url: 'my-path'});

// subscribe to ReplaySubject instead
// logged: 'my path'
shareWithReplay.subscribe(console.log);
```


In fact, if we dig into the source code we can see a very similar technique is being used. When a subscription is made, `shareReplay` will subscribe to the source, sending values through an internal `ReplaySubject` :

([source](#))

```
    return function shareReplayOperation(this: Subscriber<T>, source: Observable<T>) {
        refCount++;
        if (!subject || hasError) {
            hasError = false;
            subject = new ReplaySubject<T>(bufferSize, windowTime, scheduler);
            subscription = source.subscribe({
                next(value) { subject.next(value); },
                error(err) {
                    hasError = true;
                    subject.error(err);
                },
                complete() {
                    isComplete = true;
                    subject.complete();
                },
            });
        }

        const innerSub = subject.subscribe(this);


        return () => {
            refCount--;
            innerSub.unsubscribe();
            if (subscription && refCount === 0 && isComplete) {
                subscription.unsubscribe();
            }
        };
    };
}
```

 Ultimate Courses™

Fast-track your **RxJS skills** to build **any project or get hired** at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: Multiple subscribers sharing source

([Stackblitz](#))

```
// RxJS v6+
import { Subject, ReplaySubject } from 'rxjs';
import { pluck, share, shareReplay, tap } from 'rxjs/operators';

// simulate url change with subject
const routeEnd = new Subject<{data: any, url: string}>();
// grab url and share with subscribers
const lastUrl = routeEnd.pipe(
  tap(_ => console.log('executed')),
  pluck('url'),
  // defaults to all values so we set it to just keep and replay
  // last one
  shareReplay(1)
);
// requires initial subscription
const initialSubscriber = lastUrl.subscribe(console.log)
// simulate route change
// logged: 'executed', 'my-path'
routeEnd.next({data: {}, url: 'my-path'});
// logged: 'my-path'
const lateSubscriber = lastUrl.subscribe(console.log);
```


Additional Resources

- [shareReplay](#)  - Official docs
-

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/shareReplay.ts>

Filtering Operators

In a [push based approach](#), picking and choosing how and when to accept items is important. These operators provide techniques for accepting values from an observable source and dealing with [backpressure](#).

Contents

- [audit](#)
- [auditTime](#)
- [debounce](#)
- [debounceTime](#) ★
- [distinct](#)
- [distinctUntilChanged](#) ★
- [distinctUntilKeyChanged](#)
- [filter](#) ★
- [find](#)
- [first](#)
- [ignoreElements](#)
- [last](#)
- [sample](#)
- [single](#)
- [skip](#)
- [skipUntil](#)
- [skipWhile](#)
- [take](#) ★
- [takeLast](#)
- [takeUntil](#) ★
- [takeWhile](#)
- [throttle](#)
- [throttleTime](#)

★ - *commonly used*


audit

signature: `audit(durationSelector: (value) => Observable | Promise): Observable`

Ignore for time based on provided observable, then emit most recent value

[Examples Coming Soon!]

Additional Resources

- [audit](#)  - Official docs



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/audit.ts>

auditTime

signature: `auditTime(duration: number, scheduler?: Scheduler): Observable`

Ignore for given time then emit most recent value

Why use `auditTime`

When you are interested in ignoring a source observable for a given amount of time, you can use `auditTime`. A possible use case is to only emit certain events (i.e. mouse clicks) at a maximum rate per second. After the specified duration has passed, the timer is disabled and the most recent source value is emitted on the output Observable, and this process repeats for the next source value.

💡 If you want the timer to reset whenever a new event occurs on the source observable, you can use [debounceTime](#)


Examples

Example 1: Emit clicks at a rate of at most one click per second


([stackBlitz](#))

```
import { fromEvent } from 'rxjs';
import { auditTime } from 'rxjs/operators';

// Create observable that emits click events
const source = fromEvent(document, 'click');
// Emit clicks at a rate of at most one click per second
const example = source.pipe(auditTime(1000))
// Output (example): '(1s) --- Clicked --- (1s) --- Clicked'
const subscribe = example.subscribe(val => console.log('Clicked'))
);
```



Additional Resources

- [auditTime](#)  - Official docs
- [Time based operators comparison](#)

 Source Code:


[https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/auditTime](https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/auditTime.ts)
[.ts](#)

debounce

signature: `debounce(durationSelector: function): Observable`

Discard emitted values that take less than the specified time, based on selector function, between output.


💡 Though not as widely used as [debounceTime](#), **debounce** is important when the debounce rate is variable!

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

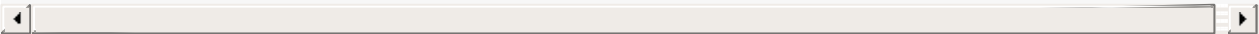
Example 1: Debounce on timer

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of, timer } from 'rxjs';
import { debounce } from 'rxjs/operators';

//emit four strings
const example = of('WAIT', 'ONE', 'SECOND', 'Last will display');

/*
    Only emit values after a second has passed between the last
    emission,
    throw away all other values
*/
const debouncedExample = example.pipe(debounce(() => timer(1000))
);
/*
    In this example, all values but the last will be omitted
    output: 'Last will display'
*/
const subscribe = debouncedExample.subscribe(val => console.log(
val));
```



Example 2: Debounce at increasing interval

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { interval, timer } from 'rxjs';
import { debounce } from 'rxjs/operators';

//emit value every 1 second, ex. 0...1...2
const interval$ = interval(1000);
//raise the debounce time by 200ms each second
const debouncedInterval = interval$.pipe(debounce(val => timer(val * 200)));
/*
  After 5 seconds, debounce time will be greater than interval time,
  all future values will be thrown away
  output: 0...1...2...3...4.....(debounce time over 1s, no values emitted)
*/
const subscribe = debouncedInterval.subscribe(val =>
  console.log(`Example Two: ${val}`)
);
```

Additional Resources

- [debounce](#) 📖 - Official docs
- [Transformation operator: debounce and debounceTime](#) 📖 💡 - André Staltz

📁 Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/debounce.ts>

debounceTime

signature: `debounceTime(dueTime: number, scheduler: Scheduler): Observable`

Discard emitted values that take less than the specified time between output


💡 This operator is popular in scenarios such as type-ahead where the rate of user input must be controlled!

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: Debouncing based on time between input

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { debounceTime, map } from 'rxjs/operators';

// elem ref
const searchBox = document.getElementById('search');

// streams
const keyup$ = fromEvent(searchBox, 'keyup');

// wait .5s between keyups to emit current value
keyup$
  .pipe(
    map((i: any) => i.currentTarget.value),
    debounceTime(500)
  )
  .subscribe(console.log);
```

Related Recipes

- [Save Indicator](#)
- [Type Ahead](#)

Additional Resources

- [debounceTime](#) 📖 - Official docs
- [Transformation operator: debounce and debounceTime](#) 📺 💰 - André Staltz
- [Time based operators comparison](#)
- [Build your own debounceTime operator](#) 📺 - Kwinten Pisman


📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/debounceTime.ts>

distinct

signature: `distinct(keySelector?, flushes?): Observable`


Emits items emitted that are distinct based on any previously emitted item.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: distinct without selector

([StackBlitz](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { distinct } from 'rxjs/operators';

of(1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
  .pipe(distinct())
  // OUTPUT: 1,2,3,4,5
  .subscribe(console.log);
```

Example 2: distinct with key selector

([StackBlitz](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { distinct } from 'rxjs/operators';

const obj1 = { id: 3, name: 'name 1' };
const obj2 = { id: 4, name: 'name 2' };
const obj3 = { id: 3, name: 'name 3' };
const vals = [obj1, obj2, obj3];

from(vals)
  .pipe(distinct(e => e.id))
  .subscribe(console.log);

/*
OUTPUT:
{id: 3, name: "name 1"}
{id: 4, name: "name 2"}
*/
```

Additional Resources

- [distinct](#)  - Official docs

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/distinct.ts>

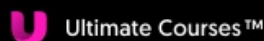
distinctUntilChanged

signature: `distinctUntilChanged(compare: function): Observable`

Only emit when the current value is different than the last.

💡 `distinctUntilChanged` uses `===` comparison by default, object references must match!

💡 If you want to compare based on an object property, you can use `distinctUntilKeyChanged` instead!



Fast-track your **RxJS skills to build any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: `distinctUntilChanged` with basic values

([StackBlitz](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { distinctUntilChanged } from 'rxjs/operators';

// only output distinct values, based on the last emitted value
const source$ = from([1, 1, 2, 2, 3, 3]);

source$
  .pipe(distinctUntilChanged())
  // output: 1,2,3
  .subscribe(console.log);
```

Example 2: distinctUntilChanged with objects

([StackBlitz](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { distinctUntilChanged } from 'rxjs/operators';

const sampleObject = { name: 'Test' };

//Objects must be same reference
const source$ = from([sampleObject, sampleObject, sampleObject]);

// only emit distinct objects, based on last emitted value
source$
  .pipe(distinctUntilChanged())
  // output: {name: 'Test'}
  .subscribe(console.log);
```

Example 3: Using custom comparer function

([StackBlitz](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { distinctUntilChanged } from 'rxjs/operators';

// only output distinct values, based on the last emitted value
const source$ = from([
  { name: 'Brian' },
  { name: 'Joe' },
  { name: 'Joe' },
  { name: 'Sue' }
]);

source$
  // custom compare for name
  .pipe(distinctUntilChanged((prev, curr) => prev.name === curr.name))
  // output: { name: 'Brian' }, { name: 'Joe' }, { name: 'Sue' }
  .subscribe(console.log);
```

Related Recipes

- [Lockscreen](#)
- [Save Indicator](#)
- [Type Ahead](#)

Additional Resources

- [distinctUntilChanged](#) 📄 - Official docs
- [Filtering operator: distinct and distinctUntilChanged](#) 🖥️💻 - André Staltz


📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/distinctUntilChanged.ts>

distinctUntilKeyChanged

signature: `distinctUntilKeyChanged(key, compare: fn): Observable`


Only emit when the specified key value has changed

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Compare based on key

([StackBlitz](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { distinctUntilKeyChanged } from 'rxjs/operators';

// only output distinct values, based on the last emitted value
const source$ = from([
  { name: 'Brian' },
  { name: 'Joe' },
  { name: 'Joe' },
  { name: 'Sue' }
]);

source$
  // custom compare based on name property
  .pipe(distinctUntilKeyChanged('name'))
  // output: { name: 'Brian' }, { name: 'Joe' }, { name: 'Sue' }
  .subscribe(console.log);
```

Example 2: Keyboard events

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { distinctUntilKeyChanged, pluck } from 'rxjs/operators';

const keys$ = fromEvent(document, 'keyup')
  .pipe(
    distinctUntilKeyChanged<KeyboardEvent>('code'),
    pluck('key')
  );

keys$.subscribe(console.log);
```

Additional Resources

- [distinctUntilKeyChanged](#)  - Official docs



Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/distinctUntilKeyChanged.ts>

filter

signature: `filter(select: Function, thisArg: any): Observable`

Emit values that pass the provided condition.


💡 If you would like to complete an observable when a condition fails, check out [takeWhile!](#)

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: filter for even numbers

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { filter } from 'rxjs/operators';

//emit (1,2,3,4,5)
const source = from([1, 2, 3, 4, 5]);
//filter out non-even numbers
const example = source.pipe(filter(num => num % 2 === 0));
//output: "Even number: 2", "Even number: 4"
const subscribe = example.subscribe(val => console.log(`Even number: ${val}`));
```

Example 2: filter objects based on property

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { filter } from 'rxjs/operators';

//emit ({name: 'Joe', age: 31}, {name: 'Bob', age:25})
const source = from([ { name: 'Joe', age: 31 }, { name: 'Bob', age : 25 }]);
//filter out people with age under 30
const example = source.pipe(filter(person => person.age >= 30));
//output: "Over 30: Joe"
const subscribe = example.subscribe(val => console.log(`Over 30: ${val.name}`));
```

Example 3: filter for number greater than specified value

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { filter } from 'rxjs/operators';

//emit every second
const source = interval(1000);
//filter out all values until interval is greater than 5
const example = source.pipe(filter(num => num > 5));
/*
  "Number greater than 5: 6"
  "Number greater than 5: 7"
  "Number greater than 5: 8"
  "Number greater than 5: 9"
*/
const subscribe = example.subscribe(val =>
  console.log(`Number greater than 5: ${val}`)
);
```

Related Recipes

- [Battleship Game](#)
- [HTTP Polling](#)
- [Game Loop](#)
- [Lockscreen](#)
- [Mine Sweeper Game](#)
- [Save Indicator](#)

Additional Resources

- [filter](#) 📖 - Official docs
- [Adding conditional logic with filter](#) 📖 💰 - John Linquist
- [Filtering operator: filter](#) 📖 💰 - André Staltz
- [Build your own filter operator](#) 📖 - Kwinten Pisman



Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/filter.ts>

find

signature: `find(predicate: function)`

Emit the first item that passes predicate then complete.


💡 If you always want the first item emitted, regardless of condition, try `first()` !

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Find click inside box, repeat when a click occurs outside of box

([StackBlitz](#))


```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { find, repeatWhen, mapTo, startWith, filter } from 'rxjs/operators';

// elem ref
const status = document.getElementById('status');

// streams
const clicks$ = fromEvent(document, 'click');

clicks$
  .pipe(
    find((event: any) => event.target.id === 'box'),
    mapTo('Found!'),
    startWith('Find me!'),
    // reset when click outside box
    repeatWhen(() =>
      clicks$.pipe(filter((event: any) => event.target.id !== 'box'))
    )
  )
  .subscribe(message => (status.innerHTML = message));
```

Additional Resources

- [find](#)  - Official docs

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/find.ts>


first

signature: `first(predicate: function, select: function)`

Emit the first value or first to pass provided expression.

💡 The counterpart to first is **last!**


💡 `first` will deliver an `EmptyError` to the Observer's error callback if the Observable completes before any next notification was sent. If you don't want this behavior, use `take(1)` instead.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

([example tests](#))

Example 1: First value from sequence

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { first } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5]);
//no arguments, emit first value
const example = source.pipe(first());
//output: "First value: 1"
const subscribe = example.subscribe(val => console.log(`First value: ${val}`));
```

Example 2: First value to pass predicate

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { first } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5]);
//emit first item to pass test
const example = source.pipe(first(num => num === 5));
//output: "First to pass test: 5"
const subscribe = example.subscribe(val =>
  console.log(`First to pass test: ${val}`)
);
```

Example 3: Utilizing default value

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { first } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5]);
//no value will pass, emit default
const example = source.pipe(first(val => val > 5, 'Nothing'));
//output: 'Nothing'
const subscribe = example.subscribe(val => console.log(val));
```

Additional Resources

- [first](#) 📖 - Official docs
- [Filtering operator: take, first, skip](#) 📺 📄 - André Staltz

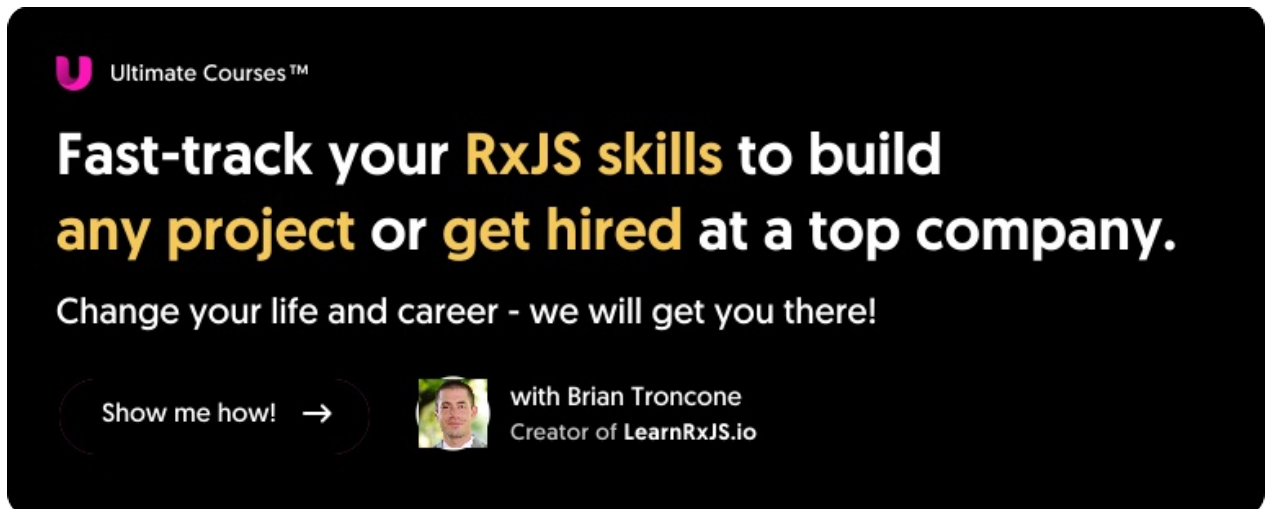
📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/first.ts>

ignoreElements

signature: `ignoreElements(): Observable`

Ignore everything but complete and error.




U Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Ignore all elements from source

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { take, ignoreElements } from 'rxjs/operators';

//emit value every 100ms
const source = interval(100);
//ignore everything but complete
const example = source.pipe(
  take(5),
  ignoreElements()
);
//output: "COMPLETE!"
const subscribe = example.subscribe(
  val => console.log(`NEXT: ${val}`),
  val => console.log(`ERROR: ${val}`),
  () => console.log('COMPLETE!')
);
```

Example 2: Only displaying error

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, throwError, of } from 'rxjs';
import { mergeMap, ignoreElements } from 'rxjs/operators';

//emit value every 100ms
const source = interval(100);
//ignore everything but error
const error = source.pipe(
  mergeMap(val => {
    if (val === 4) {
      return throwError(`ERROR AT ${val}`);
    }
    return of(val);
  }),
  ignoreElements()
);
//output: "ERROR: ERROR AT 4"
const subscribe = error.subscribe(
  val => console.log(`NEXT: ${val}`),
  val => console.log(`ERROR: ${val}`),
  () => console.log('SECOND COMPLETE!')
);
```

Additional Resources

- [ignoreElements](#)  - Official docs

 Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/ignoreElements.ts>

last

signature: `last(predicate: function): Observable`

Emit the last value emitted from source on completion, based on provided expression.

💡 The counterpart to last is **first!**

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Last value in sequence

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { from } from 'rxjs';
import { last } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5]);
//no arguments, emit last value
const example = source.pipe(last());
//output: "Last value: 5"
const subscribe = example.subscribe(val => console.log(`Last value: ${val}`));
```

Example 2: Last value to pass predicate

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { last } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5]);
//emit last even number
const exampleTwo = source.pipe(last(num => num % 2 === 0));
//output: "Last to pass test: 4"
const subscribeTwo = exampleTwo.subscribe(val =>
  console.log(`Last to pass test: ${val}`)
);
```

Example 3: Last with default value

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { last } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5]);
//no values will pass given predicate, emit default
const exampleTwo = source.pipe(last(v => v > 5, 'Nothing!'));
//output: 'Nothing!'
const subscribeTwo = exampleTwo.subscribe(val => console.log(val));
```

Additional Resources

- [last](#) 📖 - Official docs
- [Filtering operator: takeLast, last](#) 📺 📄 - André Staltz


📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/last.ts>

sample

signature: `sample(sampler: Observable): Observable`


Sample from source when provided observable emits.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: Sample source every 2 seconds

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { sample } from 'rxjs/operators';

//emit value every 1s
const source = interval(1000);
//sample last emitted value from source every 2s
const example = source.pipe(sample(interval(2000)));
//output: 2..4..6..8..
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Sample source when interval emits

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, zip, from } from 'rxjs';
import { sample } from 'rxjs/operators';

const source = zip(
  //emit 'Joe', 'Frank' and 'Bob' in sequence
  from(['Joe', 'Frank', 'Bob']),
  //emit value every 2s
  interval(2000)
);
//sample last emitted value from source every 2.5s
const example = source.pipe(sample(interval(2500)));
//output: ["Joe", 0]...["Frank", 1].....
const subscribe = example.subscribe(val => console.log(val));
```

Example 3: Distinguish between drag and click

From [Stack Overflow](#) By [Dorus](#)

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { fromEvent, merge } from 'rxjs';
import { sample, mapTo } from 'rxjs/operators';

const listener = merge(
  fromEvent(document, 'mousedown').pipe(mapTo(false)),
  fromEvent(document, 'mousemove').pipe(mapTo(true))
)
.pipe(sample(fromEvent(document, 'mouseup')))
.subscribe(isDragging => {
  console.log('Were you dragging?', isDragging);
});
```

Additional Resources

- [sample](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/sample.ts>

single

signature: `single(a: Function): Observable`


Emit single item that passes expression.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Emit first number passing predicate

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { single } from 'rxjs/operators';

//emit (1,2,3,4,5)
const source = from([1, 2, 3, 4, 5]);
//emit one item that matches predicate
const example = source.pipe(single(val => val === 4));
//output: 4
const subscribe = example.subscribe(val => console.log(val));
```

Additional Resources

- [single](#)  - Official docs

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/single.ts>

skip


signature: `skip(the: Number): Observable`

Skip the provided number of emitted values.

Why use `skip` ?

Skip allows you to ignore the first x emissions from the source. Generally `skip` is used when you have an observable that always emits certain values on subscription that you wish to ignore. Perhaps those first few aren't needed or you are subscribing to a `Replay` or `BehaviorSubject` and do not need to act on the initial values. Reach for `skip` if you are only concerned about later emissions.


You could mimic `skip` by using `filter` with indexes. Ex. `.filter((val, index) => index > 1)`

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: Skipping values before emission

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { interval } from 'rxjs';
import { skip } from 'rxjs/operators';

//emit every 1s
const source = interval(1000);
//skip the first 5 emitted values
const example = source.pipe(skip(5));
//output: 5...6...7...8.....
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Short hand for a specific filter use case

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { skip, filter } from 'rxjs/operators';




const numArrayObs = from([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]);

// 3,4,5...
const skipObs = numArrayObs.pipe(skip(2)).subscribe(console.log);

// 3,4,5...
const filterObs = numArrayObs
  .pipe(filter((val, index) => index > 1))
  .subscribe(console.log);

//Same output!
```

Additional Resources

- [skip](#)  - Official docs
- [Filtering operator: take, first, skip](#)   - André Staltz




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/skip.ts>

skipUntil

signature: `skipUntil(the: Observable): Observable`


Skip emitted values from source until provided observable emits.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Skip until observable emits

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, timer } from 'rxjs';
import { skipUntil } from 'rxjs/operators';

//emit every 1s
const source = interval(1000);
//skip emitted values from source until inner observable emits (
6s)
const example = source.pipe(skipUntil(timer(6000)));
//output: 5...6...7...8.....
const subscribe = example.subscribe(val => console.log(val));
```

Additional Resources

- [skipUntil](#)  - Official docs
-




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/skipUntil.ts>

skipWhile

signature: `skipWhile(predicate: Function): Observable`


Skip emitted values from source until provided expression is false.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Skip while values below threshold

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { skipWhile } from 'rxjs/operators';

//emit every 1s
const source = interval(1000);
//skip emitted values from source while value is less than 5
const example = source.pipe(skipWhile(val => val < 5));
//output: 5...6...7...8.....
const subscribe = example.subscribe(val => console.log(val));
```

Additional Resources

- [skipWhile](#)  - Official docs



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/skipWhile.ts>

take

signature: `take(count: number): Observable`

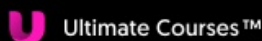
Emit provided number of values before completing.

Why use `take` ?

When you are interested in only the first emission, you want to use `take` . Maybe you want to see what the user first clicked on when they entered the page, or you would want to subscribe to the click event and just take the first emission. Another use-case is when you need to take a snapshot of data at a particular point in time but do not require further emissions. For example, a stream of user token updates, or a route guard based on a stream in an Angular application.

💡 If you want to take a variable number of values based on some logic, or another observable, you can use `takeUntil` or `takeWhile`!

💡 `take` is the opposite of `skip` where `take` will take the first n number of emissions while `skip` will skip the first n number of emissions.



**Fast-track your `RxJS` skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of `LearnRxJS.io`

Examples

Example 1: Take 1 value from source

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { take } from 'rxjs/operators';

//emit 1,2,3,4,5
const source = of(1, 2, 3, 4, 5);
//take the first emitted value then complete
const example = source.pipe(take(1));
//output: 1
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Take the first 5 values from source

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { take } from 'rxjs/operators';

//emit value every 1s
const interval$ = interval(1000);
//take the first 5 emitted values
const example = interval$.pipe(take(5));
//output: 0,1,2,3,4
const subscribe = example.subscribe(val => console.log(val));
```

Example 3: Taking first click location

([StackBlitz](#) | [jsFiddle](#))

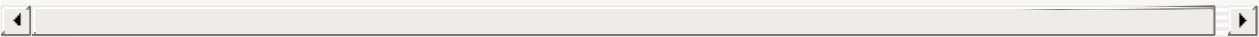
```
<div id="locationDisplay">
  Where would you click first?
</div>
```



```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { take, tap } from 'rxjs/operators';

const oneClickEvent = fromEvent(document, 'click').pipe(
  take(1),
  tap(v => {
    document.getElementById(
      'locationDisplay'
    ).innerHTML = `Your first click was on location ${v.screenX}:
    ${v.screenY}`;
  })
);

const subscribe = oneClickEvent.subscribe();
```



Related Recipes

- [Battleship Game](#)
- [Memory Game](#)

Additional Resources

- [take](#) 📖 - Official docs
- [Filtering operator: take, first, skip](#) 📺 📄 - André Staltz
- [Build your own take operator](#) 📺 - Kwinten Pisman

 Source Code:

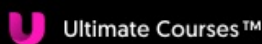
<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/take.ts>

takeLast

signature: `takeLast(count: number): Observable`

Emit the last n emitted values before completion

💡 If you want only the last emission from multiple observables, on completion of multiple observables, try [forkJoin!](#)



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: take the last 2 emitted values before completion

([StackBlitz](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { takeLast } from 'rxjs/operators';

const source = of('Ignore', 'Ignore', 'Hello', 'World!');
// take the last 2 emitted values
const example = source.pipe(takeLast(2));
// Hello, World!
const subscribe = example.subscribe(val => console.log(val));
```

Additional Resources

- [takeLast](#)  - Official docs

 Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/takeLast.ts>

takeUntil

signature: `takeUntil(notifier: Observable): Observable`

Emit values until provided observable emits.


💡 If you only need a specific number of values, try [take](#)!

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Take values until timer emits

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, timer } from 'rxjs';
import { takeUntil } from 'rxjs/operators';

//emit value every 1s
const source = interval(1000);
//after 5 seconds, emit value
const timer$ = timer(5000);
//when timer emits after 5s, complete source
const example = source.pipe(takeUntil(timer$));
//output: 0,1,2,3
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Take the first 5 even numbers

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs/observable/interval';
import { takeUntil, filter, scan, map, withLatestFrom } from 'rxjs/operators';

//emit value every 1s
const source = interval(1000);
//is number even?
const isEven = val => val % 2 === 0;
//only allow values that are even
const evenSource = source.pipe(filter(isEven));
//keep a running total of the number of even numbers out
const evenNumberCount = evenSource.pipe(scan((acc, _) => acc + 1, 0));
//do not emit until 5 even numbers have been emitted
const fiveEvenNumbers = evenNumberCount.pipe(filter(val => val > 5));

const example = evenSource.pipe(
  //also give me the current even number count for display
  withLatestFrom(evenNumberCount),
  map(([val, count]) => `Even number (${count}) : ${val}`),
  //when five even numbers have been emitted, complete source observable
  takeUntil(fiveEvenNumbers)
);
/*
    Even number (1) : 0,
    Even number (2) : 2
    Even number (3) : 4
    Even number (4) : 6
    Even number (5) : 8
*/
const subscribe = example.subscribe(val => console.log(val));
```

Example 3: Take mouse events on mouse down until mouse up

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { takeUntil, mergeMap, map } from 'rxjs/operators';

const mousedown$ = fromEvent(document, 'mousedown');
const mouseup$ = fromEvent(document, 'mouseup');
const mousemove$ = fromEvent(document, 'mousemove');

// after mousedown, take position until mouse up
mousedown$
  .pipe(
    mergeMap(_ => {
      return mousemove$.pipe(
        map((e: any) => ({
          x: e.clientX,
          y: e.clientY
        })),
        // complete inner observable on mouseup event
        takeUntil(mouseup$)
      );
    })
  )
  .subscribe(console.log);
```

Related Recipes

- [Lockscreen](#)
- [Space Invaders Game](#)
- [Swipe To Refresh](#)

Additional Resources

- [takeUntil](#) 📖 - Official docs
- [Avoiding takeUntil leaks](#) - Angular in Depth
- [Stopping a stream with takeUntil](#) 🖥️ 📄 - John Linquist
- [Build your own takeUntil operator](#) 🖥️ - Kwinten Pisman



Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/takeUntil.ts>

takeWhile

signature: `takeWhile(predicate: function(value, index): boolean, inclusive?: boolean): Observable`

Emit values until provided expression is false.


💡 When the optional `inclusive` parameter is set to `true` it will also emit the first item that didn't pass the predicate.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Take values under limit

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { takeWhile } from 'rxjs/operators';

//emit 1,2,3,4,5
const source$ = of(1, 2, 3, 4, 5);

//allow values until value from source is greater than 4, then complete
source$
  .pipe(takeWhile(val => val <= 4))
  // log: 1,2,3,4
  .subscribe(val => console.log(val));
```

Example 2: (v6.4+) takeWhile with inclusive flag

([StackBlitz](#))

```
// RxJS v6.4+
import { of } from 'rxjs';
import { takeWhile, filter } from 'rxjs/operators';

const source$ = of(1, 2, 3, 9);

source$
  // with inclusive flag, the value causing the predicate to return false will also be emitted
  .pipe(takeWhile(val => val <= 3, true))
  // log: 1, 2, 3, 9
  .subscribe(console.log);
```

Example 3: Difference between takeWhile and filter

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { takeWhile, filter } from 'rxjs/operators';

// emit 3, 3, 3, 9, 1, 4, 5, 8, 96, 3, 66, 3, 3, 3
const source$ = of(3, 3, 3, 9, 1, 4, 5, 8, 96, 3, 66, 3, 3, 3);

// allow values until value from source equals 3, then complete
source$
  .pipe(takeWhile(it => it === 3))
  // log: 3, 3, 3
  .subscribe(val => console.log('takeWhile', val));

source$
  .pipe(filter(it => it === 3))
  // log: 3, 3, 3, 3, 3, 3, 3
  .subscribe(val => console.log('filter', val));
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Catch The Dot Game](#)
- [Click Ninja Game](#)
- [Flappy Bird Game](#)
- [Mine Sweeper Game](#)
- [Platform Jumper Game](#)
- [Smart Counter](#)
- [Swipe To Refresh](#)
- [Tetris Game](#)

Additional Resources

- [takeWhile](#)  - Official docs
- [Completing a stream with takeWhile](#)   - John Linquist




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/takeWhile.ts>

throttle

signature: `throttle(durationSelector: function(value): Observable | Promise): Observable`


Emit value on the leading edge of an interval, but suppress new values until `durationSelector` has completed.

 Ultimate Courses™

Fast-track your **RxJS skills to build **any project** or **get hired** at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Throttle for 2 seconds, based on second observable

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { throttle } from 'rxjs/operators';

//emit value every 1 second
const source = interval(1000);
//throttle for 2 seconds, emit latest value
const example = source.pipe(throttle(val => interval(2000)));
//output: 0...3...6...9
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Throttle with promise




([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { throttle, map } from 'rxjs/operators';

//emit value every 1 second
const source = interval(1000);
//incrementally increase the time to resolve based on source
const promise = val =>
  new Promise(resolve =>
    setTimeout(() => resolve(`Resolved: ${val}`), val * 100)
  );
//when promise resolves emit item from source
const example = source.pipe(
  throttle(promise),
  map(val => `Throttled off Promise: ${val}`)
);

const subscribe = example.subscribe(val => console.log(val));
```

Additional Resources

- [throttle](#)  - Official docs
- [Filtering operator: throttle and throttleTime](#)   - André Staltz




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/throttle.ts>

throttleTime

signature: `throttleTime(duration: number, scheduler: Scheduler, config: ThrottleConfig): Observable`


Emit first value then ignore for specified duration

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Emit first value, ignore for 5s window

([StackBlitz](#))


```
// RxJS v6+
import { interval } from 'rxjs';
import { throttleTime } from 'rxjs/operators';

// emit value every 1 second
const source = interval(1000);
/*
  emit the first value, then ignore for 5 seconds. repeat...
*/
const example = source.pipe(throttleTime(5000));
// output: 0...6...12
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Emit on trailing edge using config

([StackBlitz](#))




```
// RxJS v6+
import { interval, asyncScheduler } from 'rxjs';
import { throttleTime } from 'rxjs/operators';

const source = interval(1000);
/*
  emit the first value, then ignore for 5 seconds. repeat...
*/
const example = source.pipe(throttleTime(
  5000,
  asyncScheduler,
  { trailing: true }
));
// output: 5...11...17
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Horizontal Scroll Indicator](#)
- [Lockscreen](#)

Additional Resources

- [throttleTime](#)  - Official docs
- [Filtering operator: throttle and throttleTime](#)   - André Staltz
- [Time based operators comparison](#)

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/throttleTime.ts>

Transformation Operators

Transforming values as they pass through the operator chain is a common task. These operators provide transformation techniques for nearly any use-case you will encounter.

Contents


- [buffer](#)
- [bufferCount](#)
- [bufferTime](#) ★
- [bufferToggle](#)
- [bufferWhen](#)
- [concatMap](#) ★
- [concatMapTo](#)
- [exhaustMap](#)
- [expand](#)
- [groupBy](#)
- [map](#) ★
- [mapTo](#)
- [mergeMap / flatMap](#) ★
- [mergeScan](#)
- [partition](#)
- [pluck](#)
- [reduce](#)
- [scan](#) ★
- [switchMap](#) ★
- [switchMapTo](#)
- [toArray](#)
- [window](#)
- [windowCount](#)
- [windowTime](#)
- [windowToggle](#)
- [windowWhen](#)

★ - *commonly used*

buffer

signature: `buffer(closingNotifier: Observable): Observable`


Collect output values until provided observable emits, emit as array.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: Using buffer to recognize double clicks

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { buffer, filter, throttleTime } from 'rxjs/operators';

// streams
const clicks$ = fromEvent(document, 'click');

/*
Collect clicks that occur, after 250ms emit array of clicks
*/
clicks$
  .pipe(
    buffer(clicks$.pipe(throttleTime(250))),
    // if array is greater than 1, double click occurred
    filter(clickArray => clickArray.length > 1)
  )
  .subscribe(() => console.log('Double Click!'));
```

Example 2: Buffer until document click

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))




```
// RxJS v6+
import { interval, fromEvent } from 'rxjs';
import { buffer } from 'rxjs/operators';

//Create an observable that emits a value every second
const myInterval = interval(1000);
//Create an observable that emits every time document is clicked
const bufferBy = fromEvent(document, 'click');
/*
Collect all values emitted by our interval observable until we c
lick document. This will cause the bufferBy Observable to emit a
value, satisfying the buffer. Pass us all collected values sinc
e last buffer as an array.
*/
const myBufferedInterval = myInterval.pipe(buffer(bufferBy));
//Print values to console
//ex. output: [1,2,3] ... [4,5,6,7,8]
const subscribe = myBufferedInterval.subscribe(val =>
  console.log(' Buffered Values:', val)
);
```

Related Recipes

- [Game Loop](#)

Additional Resources

- [buffer](#)  - Official docs
- [Transformation operator: buffer](#)   - André Staltz


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/buffer.ts>

bufferCount

signature: `bufferCount(bufferSize: number, startBufferEvery: number = null): Observable`


Collect emitted values until provided number is fulfilled, emit as array.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Collect buffer and emit after specified number of values

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { interval } from 'rxjs';
import { bufferCount } from 'rxjs/operators';

//Create an observable that emits a value every second
const source = interval(1000);
//After three values are emitted, pass on as an array of buffered values
const bufferThree = source.pipe(bufferCount(3));
//Print values to console
//ex. output [0,1,2]...[3,4,5]
const subscribe = bufferThree.subscribe(val =>
  console.log('Buffered Values:', val)
);
```

Example 2: Overlapping buffers

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { bufferCount } from 'rxjs/operators';

//Create an observable that emits a value every second
const source = interval(1000);
/*
bufferCount also takes second argument, when to start the next buffer
for instance, if we have a bufferCount of 3 but second argument
(startBufferEvery) of 1:
1st interval value:
buffer 1: [0]
2nd interval value:
buffer 1: [0,1]
buffer 2: [1]
3rd interval value:
buffer 1: [0,1,2] Buffer of 3, emit buffer
buffer 2: [1,2]
buffer 3: [2]
4th interval value:
buffer 2: [1,2,3] Buffer of 3, emit buffer
buffer 3: [2, 3]
buffer 4: [3]
*/
const bufferEveryOne = source.pipe(bufferCount(3, 1));
//Print values to console
const subscribe = bufferEveryOne.subscribe(val =>
  console.log('Start Buffer Every 1:', val)
);
```

Example 3: Last n keyboard presses tracking

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent, of } from 'rxjs';
import { bufferCount, map, mergeMap, tap } from 'rxjs/operators';

const fakeKeyPressesPost = keypresses =>
  of(201).pipe(
    tap(_ => {
      console.log(`received key presses are: ${keypresses}`);
      document.getElementById('output').innerText = keypresses;
    })
  );

fromEvent(document, 'keydown')
  .pipe(
    map((e: KeyboardEvent) => e.key),
    bufferCount(5),
    mergeMap(fakeKeyPressesPost)
  )
  .subscribe();
```

Additional Resources

- [bufferCount](#)  - Official docs



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/bufferCount.ts>

bufferTime

signature: `bufferTime(bufferTimeSpan: number, bufferCreationInterval: number, scheduler: Scheduler): Observable`

Collect emitted values until provided time has passed, emit as array.

Examples

Example 1: Buffer for 2 seconds

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { bufferTime } from 'rxjs/operators';

//Create an observable that emits a value every 500ms
const source = interval(500);
//After 2 seconds have passed, emit buffered values as an array
const example = source.pipe(bufferTime(2000));
//Print values to console
//ex. output [0,1,2]...[3,4,5,6]
const subscribe = example.subscribe(val =>
  console.log('Buffered with Time:', val)
);
```


Example 2: Multiple active buffers

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { bufferTime } from 'rxjs/operators';

//Create an observable that emits a value every 500ms
const source = interval(500);
/*
bufferTime also takes second argument, when to start the next bu
ffer (time in ms)
for instance, if we have a bufferTime of 2 seconds but second ar
gument (bufferCreationInterval) of 1 second:
ex. output: [0,1,2]...[1,2,3,4,5]...[3,4,5,6,7]
*/
const example = source.pipe(bufferTime(2000, 1000));
//Print values to console
const subscribe = example.subscribe(val =>
  console.log('Start Buffer Every 1s:', val)
);
```

Additional Resources

- [bufferTime](#)  - Official docs
- [Time based operators comparison](#)


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/bufferTime.ts>

bufferToggle

signature: `bufferToggle(openings: Observable, closingSelector: Function): Observable`


Toggle on to catch emitted values from source, toggle off to emit buffered values as array.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Toggle buffer on and off at interval

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { bufferToggle } from 'rxjs/operators';

//emit value every second
const sourceInterval = interval(1000);
//start first buffer after 5s, and every 5s after
const startInterval = interval(5000);
//emit value after 3s, closing corresponding buffer
const closingInterval = val => {
  console.log(`Value ${val} emitted, starting buffer! Closing in 3s!`);
  return interval(3000);
};
//every 5s a new buffer will start, collecting emitted values for 3s then emitting buffered values
const bufferToggleInterval = sourceInterval.pipe(
  bufferToggle(startInterval, closingInterval)
);
//log to console
//ex. emitted buffers [4,5,6]...[9,10,11]
const subscribe = bufferToggleInterval.subscribe(val =>
  console.log('Emitted Buffer:', val)
);
```

Example 2: Toggle buffer on and off on mouse down/up

([StackBlitz](#))

```
import { fromEvent } from 'rxjs';
import { bufferToggle } from 'rxjs/operators';

fromEvent(document, 'mousemove')
  .pipe(
    bufferToggle(
      fromEvent(document, 'mousedown'), _ =>
        fromEvent(document, 'mouseup')
    )
  )
  .subscribe(console.log);
```

Additional Resources

- [bufferToggle](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/bufferToggle.ts>

bufferWhen

signature: `bufferWhen(closingSelector: function): Observable`


Collect all values until closing selector emits, emit buffered values.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Emit buffer based on interval

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { bufferWhen } from 'rxjs/operators';

//emit value every 1 second
const oneSecondInterval = interval(1000);
//return an observable that emits value every 5 seconds
const fiveSecondInterval = () => interval(5000);
//every five seconds, emit buffered values
const bufferWhenExample = oneSecondInterval.pipe(
  bufferWhen(fiveSecondInterval)
);
//log values
//ex. output: [0,1,2,3]...[4,5,6,7,8]
const subscribe = bufferWhenExample.subscribe(val =>
  console.log('Emitted Buffer: ', val)
);
```

Additional Resources

- [bufferWhen](#)  - Official docs

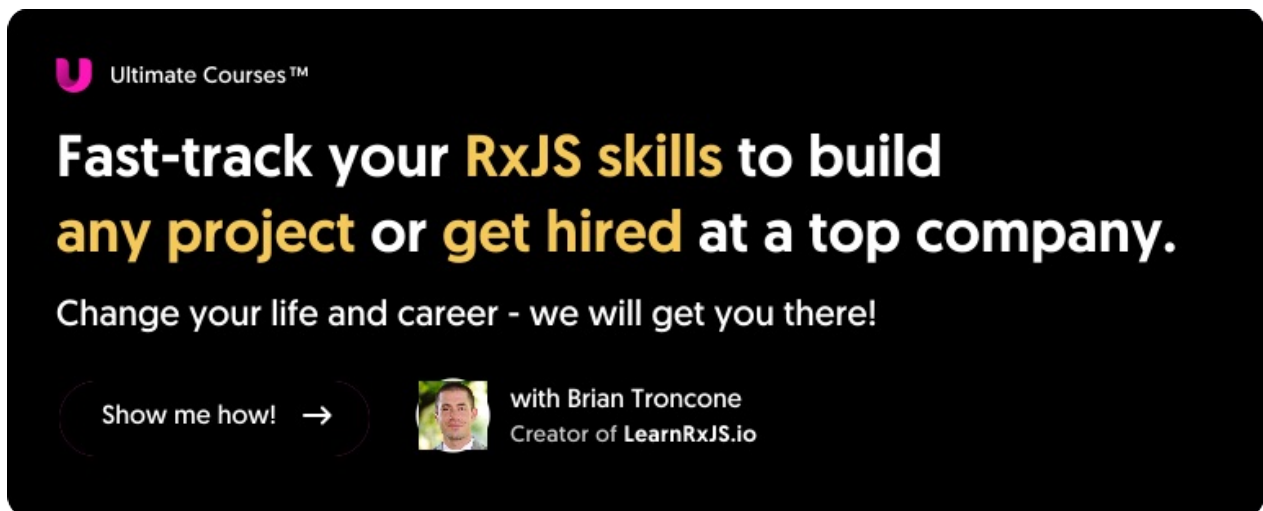
 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/bufferWhen.ts>

concatMap

signature: `concatMap(project: function, resultSelector: function): Observable`

Map values to inner observable, subscribe and emit in order.



U Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →

with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Demonstrating the difference between `concatMap` and `mergeMap`

([StackBlitz](#))

💡 Note the difference between `concatMap` and `mergeMap`. Because `concatMap` does not subscribe to the next observable until the previous completes, the value from the source delayed by 2000ms will be emitted first. Contrast this with `mergeMap` which subscribes immediately to inner observables, the observable with the lesser delay (1000ms) will emit, followed by the observable which takes 2000ms to complete.

```
// RxJS v6+
import { of } from 'rxjs';
import { concatMap, delay, mergeMap } from 'rxjs/operators';

//emit delay value
const source = of(2000, 1000);
// map value from source into inner observable, when complete emit result and move to next
const example = source.pipe(
  concatMap(val => of(`Delayed by: ${val}ms`).pipe(delay(val)))
);
//output: With concatMap: Delayed by: 2000ms, With concatMap: Delayed by: 1000ms
const subscribe = example.subscribe(val =>
  console.log(`With concatMap: ${val}`)
);

// showing the difference between concatMap and mergeMap
const mergeMapExample = source
  .pipe(
    // just so we can log this after the first example has run
    delay(5000),
    mergeMap(val => of(`Delayed by: ${val}ms`).pipe(delay(val)))
  )
  .subscribe(val => console.log(`With mergeMap: ${val}`));
```

Example 2: Map to promise

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { concatMap } from 'rxjs/operators';

//emit 'Hello' and 'Goodbye'
const source = of('Hello', 'Goodbye');
//example with promise
const examplePromise = val => new Promise(resolve => resolve(`${
val} World!`));
// map value from source into inner observable, when complete em
it result and move to next
const example = source.pipe(concatMap(val => examplePromise(val))
);
//output: 'Example w/ Promise: 'Hello World', Example w/ Promise
: 'Goodbye World'
const subscribe = example.subscribe(val =>
  console.log('Example w/ Promise:', val)
);
```





Example 3: Supplying a projection function

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { concatMap } from 'rxjs/operators';

//emit 'Hello' and 'Goodbye'
const source = of('Hello', 'Goodbye');
//example with promise
const examplePromise = val => new Promise(resolve => resolve(`${
val} World!`));
//result of first param passed to second param selector function
before being returned
const example = source.pipe(
  concatMap(val => examplePromise(val), result => `${result} w/
selector!`)
);
//output: 'Example w/ Selector: 'Hello w/ Selector', Example w/
Selector: 'Goodbye w/ Selector'
const subscribe = example.subscribe(val =>
  console.log('Example w/ Selector:', val)
);
```

Additional Resources

- [concatMap](#)  - Official docs
- [Use RxJS concatMap to map and concat higher order observables](#)   - André Staltz
- [Build your own concatMap operator](#)  - Kwinten Pisman

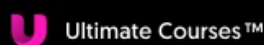
 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/concatMap.ts>

concatMapTo

signature: `concatMapTo(observable: Observable, resultSelector: function): Observable`

Subscribe to provided observable when previous completes, emit values.



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Map to basic observable (simulating request)

([StackBlitz](#))

```
// RxJS v6+
import { of, interval } from 'rxjs';
import { concatMapTo, delay, take } from 'rxjs/operators';

//emit value every 2 seconds
const sampleInterval = interval(500).pipe(take(5));
const fakeRequest = of('Network request complete').pipe(delay(3000));
//wait for first to complete before next is subscribed
const example = sampleInterval.pipe(concatMapTo(fakeRequest));
//result
//output: Network request complete...3s...Network request complete'
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Using projection with `concatMap`

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { interval } from 'rxjs';
import { concatMapTo, take } from 'rxjs/operators';
//emit value every 2 seconds
const interval$ = interval(2000);
//emit value every second for 5 seconds
const source = interval(1000).pipe(take(5));
/*
    ***Be Careful***: In situations like this where the source emits
    at a faster pace
    than the inner observable completes, memory issues can arise.
    (interval emits every 1 second, basicTimer completes every 5)
*/
// basicTimer will complete after 5 seconds, emitting 0,1,2,3,4
const example = interval$.pipe(
  concatMapTo(
    source,
    (firstInterval, secondInterval) => `${firstInterval} ${secondInterval}`
  )
);
/*
    output: 0 0
           0 1
           0 2
           0 3
           0 4
           1 0
           1 1
           continued...
*/
const subscribe = example.subscribe(val => console.log(val));
```

Additional Resources

- [concatMapTo](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/concatMapTo.ts>

exhaustMap

signature: `exhaustMap(project: function, resultSelector: function): Observable`


Map to inner observable, ignore other values until that observable completes.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: exhaustMap with interval

([Stackblitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, merge, of } from 'rxjs';
import { delay, take, exhaustMap } from 'rxjs/operators';

const sourceInterval = interval(1000);
const delayedInterval = sourceInterval.pipe(
  delay(10),
  take(4)
);

const exhaustSub = merge(
  // delay 10ms, then start interval emitting 4 values
  delayedInterval,
  // emit immediately
  of(true)
)
.pipe(exhaustMap(_ => sourceInterval.pipe(take(5))))
/*
 * The first emitted value (of(true)) will be mapped
 * to an interval observable emitting 1 value every
 * second, completing after 5.
 * Because the emissions from the delayed interval
 * fall while this observable is still active they will be ignored.
 *
 * Contrast this with concatMap which would queue,
 * switchMap which would switch to a new inner observable each
 * emission,
 * and mergeMap which would maintain a new subscription for each
 * emitted value.
 */
// output: 0, 1, 2, 3, 4
.subscribe(val => console.log(val));
```

Example 2: Another exhaustMap with interval

([Stackblitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { exhaustMap, tap, take } from 'rxjs/operators';

const firstInterval = interval(1000).pipe(take(10));
const secondInterval = interval(1000).pipe(take(2));

const exhaustSub = firstInterval
  .pipe(
    exhaustMap(f => {
      console.log(`Emission Corrected of first interval: ${f}`);
      return secondInterval;
    })
  )
  /*
```

When we subscribed to the first interval, it starts to emit a values (starting 0).

This value is mapped to the second interval which then begins to emit (starting 0).

While the second interval is active, values from the first interval are ignored.

We can see this when firstInterval emits number 3,6, and so on...

Output:

Emission of first interval: 0

0

1

Emission of first interval: 3

0

1

Emission of first interval: 6

0

1

Emission of first interval: 9

0

1

*/

```
.subscribe(s => console.log(s));
```

Related Recipes

- [Swipe To Refresh](#)

Outside Examples

`exhaustMap` for login effect in [@ngrx example app](#)

([Source](#))

```
@Effect()
login$ = this.actions$.pipe(
  ofType(AuthActionTypes.Login),
  map((action: Login) => action.payload),
  exhaustMap((auth: Authenticate) =>
    this.authService
      .login(auth)
      .pipe(
        map(user => new LoginSuccess({ user })),
        catchError(error => of(new LoginFailure(error)))
      )
  )
);
```

Additional Resources

- [exhaustMap](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/exhaustMap.ts>

expand

signature: `expand(project: function, concurrent: number, scheduler: Scheduler): Observable`


Recursively call provided function.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or **get hired** at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Add one for each invocation

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, of } from 'rxjs';
import { expand, take } from 'rxjs/operators';

//emit 2
const source = of(2);
const example = source.pipe(
  //recursively call supplied function
  expand(val => {
    //2,3,4,5,6
    console.log(`Passed value: ${val}`);
    //3,4,5,6
    return of(1 + val);
  }),
  //call 5 times
  take(5)
);
/*
  "RESULT: 2"
  "Passed value: 2"
  "RESULT: 3"
  "Passed value: 3"
  "RESULT: 4"
  "Passed value: 4"
  "RESULT: 5"
  "Passed value: 5"
  "RESULT: 6"
  "Passed value: 6"
*/
//output: 2,3,4,5,6
const subscribe = example.subscribe(val => console.log(`RESULT:
${val}`));
```

Related Recipes

- [Game Loop](#)

Additional Resources

- [expand](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/expand.ts>

groupBy

signature: `groupBy(keySelector: Function, elementSelector: Function): Observable`


Group into observables based on provided value.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Group by property

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { groupBy, mergeMap, toArray } from 'rxjs/operators';

const people = [
  { name: 'Sue', age: 25 },
  { name: 'Joe', age: 30 },
  { name: 'Frank', age: 25 },
  { name: 'Sarah', age: 35 }
];
//emit each person
const source = from(people);
//group by age
const example = source.pipe(
  groupBy(person => person.age),
  // return each item in group as array
  mergeMap(group => group.pipe(toArray()))
);
/*
  output:
  [{age: 25, name: "Sue"},{age: 25, name: "Frank"}]
  [{age: 30, name: "Joe"}]
  [{age: 35, name: "Sarah"}]
*/
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Group by into key - values

([StackBlitz](#))

```
// RxJS v6+
import { from, of, zip } from 'rxjs';
import { groupBy, mergeMap, toArray } from 'rxjs/operators';

const people = [
  { name: 'Sue', age: 25 },
  { name: 'Joe', age: 30 },
  { name: 'Frank', age: 25 },
  { name: 'Sarah', age: 35 }
];

from(people)
  .pipe(
    groupBy(person => person.age, p => p.name),
    mergeMap(group => zip(of(group.key), group.pipe(toArray())))
  )
  .subscribe(console.log);

/*
  output:
  [25, ["Sue", "Frank"]]
  [30, ["Joe"]]
  [35, ["Sarah"]]
*/
```

Additional Resources

- [groupBy](#) 📄 - Official docs
- [Group higher order observables with RxJS groupBy](#) 🖥️ 💡 - André Staltz
- [Use groupBy in real RxJS applications](#) 🖥️ 💡 - André Staltz


📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/groupBy.ts>

map

signature: `map(project: Function, thisArg: any): Observable`


Apply projection with each value from source.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Add 10 to each number

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { map } from 'rxjs/operators';

//emit (1,2,3,4,5)
const source = from([1, 2, 3, 4, 5]);
//add 10 to each value
const example = source.pipe(map(val => val + 10));
//output: 11,12,13,14,15
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Map to single property

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { map } from 'rxjs/operators';

//emit ({name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name:
'Ryan', age: 50})
const source = from([
  { name: 'Joe', age: 30 },
  { name: 'Frank', age: 20 },
  { name: 'Ryan', age: 50 }
]);
//grab each persons name, could also use pluck for this scenario
const example = source.pipe(map(({ name }) => name));
//output: "Joe","Frank","Ryan"
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Catch The Dot Game](#)
- [Game Loop](#)
- [HTTP Polling](#)
- [Lockscreen](#)
- [Memory Game](#)
- [Mine Sweeper Game](#)
- [Save Indicator](#)
- [Smart Counter](#)
- [Space Invaders Game](#)
- [Stop Watch](#)
- [Swipe To Refresh](#)
- [Tetris Game](#)
- [Type Ahead](#)

Additional Resources

- [map](#) 📄 - Official docs
 - [map vs flatMap](#) 📄 - Ben Lesh
 - [Transformation operator: map and mapTo](#) 📄 💰 - André Staltz
 - [Build your own map operator](#) 📄 - Kwinten Pisman
-




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/map.ts>

mapTo

signature: `mapTo(value: any): Observable`


Map emissions to constant value.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Map every emission to string

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { mapTo } from 'rxjs/operators';

//emit value every two seconds
const source = interval(2000);
//map all emissions to one value
const example = source.pipe(mapTo('HELLO WORLD!'));
//output: 'HELLO WORLD!'...'HELLO WORLD!'...'HELLO WORLD!'...
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Mapping clicks to string

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { mapTo } from 'rxjs/operators';

//emit every click on document
const source = fromEvent(document, 'click');
//map all emissions to one value
const example = source.pipe(mapTo('GOODBYE WORLD!'));
//output: (click)'GOODBYE WORLD!'...
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [HTTP Polling](#)
- [Save Indicator](#)
- [Smart Counter](#)
- [Stop Watch](#)

Additional Resources

- [mapTo](#) 📄 - Official docs
- [Changing behavior with mapTo](#) 🖨️ 💰 📄 - John Linquist
- [Transformation operator: map and mapTo](#) 🖨️ 💰 📄 - André Staltz

📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/mapTo.ts>

mergeMap

signature: `mergeMap(project: function: Observable, resultSelector: function: any, concurrent: number): Observable`

Map to observable, emit values.

💡 `flatMap` is an alias for `mergeMap`!

💡 If only one inner subscription should be active at a time, try `switchMap` !

💡 If the order of emission and subscription of inner observables is important, try `concatMap` !

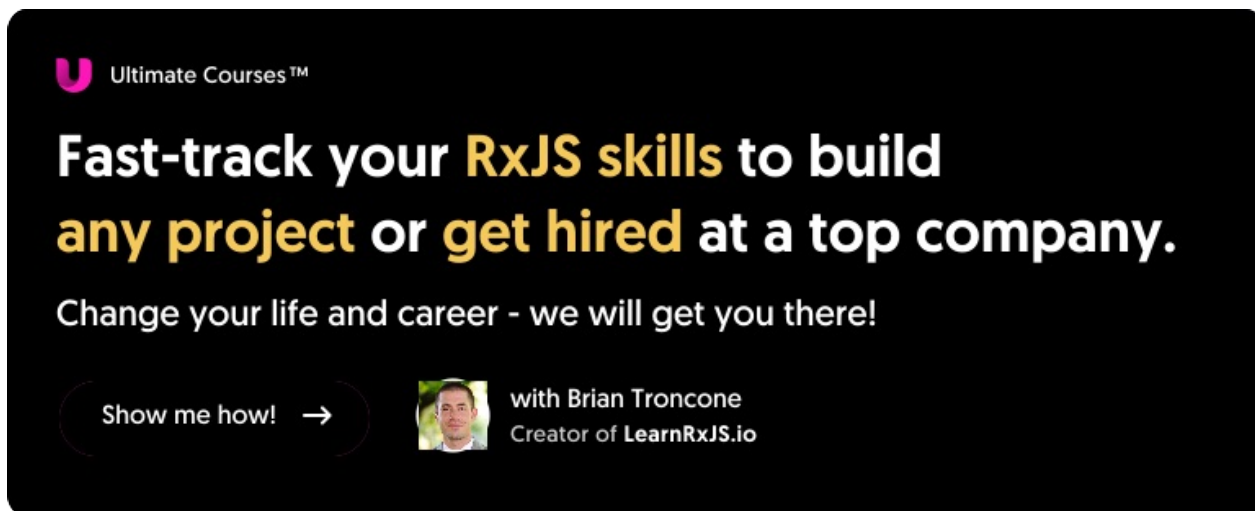
Why use `mergeMap` ?

This operator is best used when you wish to flatten an inner observable but want to manually control the number of inner subscriptions.

For instance, when using `switchMap` each inner subscription is completed when the source emits, allowing only one active inner subscription. In contrast, `mergeMap` allows for multiple inner subscriptions to be active at a time. Because of this, one of the most common use-case for `mergeMap` is requests that should not be canceled, think writes rather than reads. Note that if order must be maintained `concatMap` is a better option.

Be aware that because `mergeMap` maintains multiple active inner subscriptions at once it's possible to create a memory leak through long-lived inner subscriptions. A basic example would be if you were mapping to an observable with an inner timer, or a stream of dom events. In these cases, if you still wish to utilize `mergeMap` you may want to take advantage of another operator to


manage the completion of the inner subscription, think `take` or `takeUntil` . You can also limit the number of active inner subscriptions at a time with the `concurrent` parameter, seen in [example 5](#).



U Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →  with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: mergeMap simulating save of click locations

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent, of } from 'rxjs';
import { mergeMap, delay } from 'rxjs/operators';

// faking network request for save
const saveLocation = location => {
  return of(location).pipe(delay(500));
};

// streams
const click$ = fromEvent(document, 'click');

click$
  .pipe(
    mergeMap((e: MouseEvent) => {
      return saveLocation({
        x: e.clientX,
        y: e.clientY,
        timestamp: Date.now()
      });
    })
  )
  // Saved! {x: 98, y: 170, ...}
  .subscribe(r => console.log('Saved!', r));
```

Example 2: mergeMap with ajax observable

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { ajax } from 'rxjs/ajax';
import { mergeMap } from 'rxjs/operators';

// free api url
const API_URL = 'https://jsonplaceholder.typicode.com/todos/1';

// streams
const click$ = fromEvent(document, 'click');

click$
  .pipe(
    /*
     * Using mergeMap for example, but generally for GET request
     * you will prefer switchMap.
     * Also, if you do not need the parameter like
     * below you could use mergeMapTo instead.
     * ex. mergeMapTo(ajax.getJSON(API_URL))
     */
    mergeMap(() => ajax.getJSON(API_URL))
  )
  // { userId: 1, id: 1, ...}
  .subscribe(console.log);
```

Example 3: mergeMap with promise (could also use [from](#) to convert to observable)

([StackBlitz](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { mergeMap } from 'rxjs/operators';

// helper to create promise
const myPromise = val =>
  new Promise(resolve => resolve(`${val} World From Promise!`));

// emit 'Hello'
const source$ = of('Hello');

// map to promise and emit result
source$
  .pipe(mergeMap(val => myPromise(val)))
  // output: 'Hello World From Promise'
  .subscribe(val => console.log(val));
```

Example 4: mergeMap with resultSelector

([StackBlitz](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { mergeMap } from 'rxjs/operators';

// helper to create promise
const myPromise = val =>
  new Promise(resolve => resolve(`${val} World From Promise!`));

// emit 'Hello'
const source$ = of('Hello');

source$
  .pipe(
    mergeMap(
      val => myPromise(val),
      /*
        you can also supply a second argument which receives the s
        ource value and emitted
        value of inner observable or promise
      */
      (valueFromSource, valueFromPromise) => {
        return `Source: ${valueFromSource}, Promise: ${valueFrom
Promise}`;
      }
    )
  )
  // output: "Source: Hello, Promise: Hello World From Promise!"
  .subscribe(val => console.log(val));
```

Example 5: mergeMap with concurrent value

([StackBlitz](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { mergeMap, take } from 'rxjs/operators';

// emit value every 1s
const source$ = interval(1000);

source$
  .pipe(
    mergeMap(
      // project
      val => interval(5000).pipe(take(2)),
      // resultSelector
      (oVal, iVal, oIndex, iIndex) => [oIndex, oVal, iIndex, iVal
],
      // concurrent
      2
    )
  )
  /*
    Output:
    [0, 0, 0, 0] <--1st inner observable
    [1, 1, 0, 0] <--2nd inner observable
    [0, 0, 1, 1] <--1st inner observable
    [1, 1, 1, 1] <--2nd inner observable
    [2, 2, 0, 0] <--3rd inner observable
    [3, 3, 0, 0] <--4th inner observable
  */
  .subscribe(val => console.log(val));
```

Related Recipes

- [Breakout Game](#)
- [HTTP Polling](#)
- [Save Indicator](#)
- [Swipe To Refresh](#)

Additional Resources

- [mergeMap](#) 📄 - Official docs
- [map vs flatMap](#) 📄 💡 - Ben Lesh
- [Async requests and responses in RxJS](#) 📄 💡 - André Staltz
- [Use RxJS mergeMap to map and merge higher order observables](#) 📄 💡 - André Staltz
- [Use RxJS mergeMap for fine grain custom behavior](#) 📄 💡 - André Staltz
- [Build your own mergeMap operator](#) 📄 - Kwinten Pisman


📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/mergeMap.ts>

mergeScan

signature: `mergeScan(accumulator: (acc, value, index: number) => ObservableInput, seed, concurrent: number = Number.POSITIVE_INFINITY): OperatorFunction`


Accumulate value over time via merged observables.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Accumulate total duration mouse held down over time

([StackBlitz](#))


```
// RxJS v6+
import { fromEvent, interval } from 'rxjs';
import { mergeScan, take, takeUntil, map, scan } from 'rxjs/operators';

// reference
const durationElem = document.getElementById('duration');

// streams
const mouseDown$ = fromEvent(document, 'mousedown');
const mouseUp$ = fromEvent(document, 'mouseup');

// accumulate time mouse held down over time
mouseDown$
  .pipe(
    mergeScan((acc, curr) => {
      return interval(1000).pipe(
        scan((a, _) => ++a, 0),
        map((val: any) => val + acc),
        takeUntil(mouseUp$)
      );
    }, 0)
    // output: 1s...2s...3s...4s...
  )
  .subscribe(val => (durationElem.innerHTML = `${val}s`));
```

Additional Resources

- [pluck](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/mergeScan.ts>

partition

signature: `partition(predicate: function: boolean, thisArg: any): [Observable, Observable]`


Split one observable into two based on provided predicate.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Split even and odd numbers

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from, merge } from 'rxjs';
import { partition, map } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5, 6]);
//first value is true, second false
const [evens, odds] = source.pipe(partition(val => val % 2 === 0))
);
/*
  Output:
  "Even: 2"
  "Even: 4"
  "Even: 6"
  "Odd: 1"
  "Odd: 3"
  "Odd: 5"
*/
const subscribe = merge(
  evens.pipe(map(val => `Even: ${val}`)),
  odds.pipe(map(val => `Odd: ${val}`))
).subscribe(val => console.log(val));
```



Example 2: Split success and errors

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { merge, of, from } from 'rxjs';
import { map, partition, catchError } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5, 6]);
//if greater than 3 throw
const example = source.pipe(
  map(val => {
    if (val > 3) {
      throw `${val} greater than 3!`;
    }
    return { success: val };
  }),
  catchError(val => of({ error: val })))
);
//split on success or error
const [success, error] = example.pipe(partition(res => res.success));
/*
  Output:
  "Success! 1"
  "Success! 2"
  "Success! 3"
  "Error! 4 greater than 3!"
*/
const subscribe = merge(
  success.pipe(map(val => `Success! ${val.success}`)),
  error.pipe(map(val => `Error! ${val.error}`))
).subscribe(val => console.log(val));
```

Example 3: (v6.5+) Partition as a static function

([StackBlitz](#))

```
// RxJS v6.5+
import { merge, of, from, partition } from 'rxjs';
import { map, catchError } from 'rxjs/operators';

const source = from([1, 2, 3, 4, 5, 6]);
//if greater than 3 throw
const example = source.pipe(
  map(val => {
    if (val > 3) {
      throw `${val} greater than 3!`;
    }
    return { success: val };
  }),
  catchError(val => of({ error: val })))
);
// split on success or error
const [success, error] = partition(example, res => res.success);
/*
  Output:
  "Success! 1"
  "Success! 2"
  "Success! 3"
  "Error! 4 greater than 3!"
*/
const subscribe = merge(
  success.pipe(map(val => `Success! ${val.success}`)),
  error.pipe(map(val => `Error! ${val.error}`))
).subscribe(val => console.log(val));
```

Additional Resources

- [partition](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/observable/partition.ts>

pluck

signature: `pluck(properties: ...args): Observable`


Select properties to emit.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Pluck object property

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from } from 'rxjs';
import { pluck } from 'rxjs/operators';

const source = from([ { name: 'Joe', age: 30 }, { name: 'Sarah',
age: 35 } ]);
//grab names
const example = source.pipe(pluck('name'));
//output: "Joe", "Sarah"
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Pluck nested properties

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { from } from 'rxjs';
import { pluck } from 'rxjs/operators';

const source = from([
  { name: 'Joe', age: 30, job: { title: 'Developer', language: 'JavaScript' } },
  //will return undefined when no job is found
  { name: 'Sarah', age: 35 }
]);
//grab title property under job
const example = source.pipe(pluck('job', 'title'));
//output: "Developer" , undefined
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Breakout Game](#)
- [Car Racing Game](#)
- [Lockscreen](#)
- [Memory Game](#)
- [Mine Sweeper Game](#)
- [Platform Jumper Game](#)
- [Tetris Game](#)

Additional Resources

- [pluck](#)  - Official docs

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/pluck.ts>

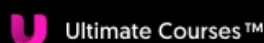
reduce

signature: `reduce(accumulator: function, seed: any): Observable`

Reduces the values from source observable to a single value that's emitted when the source completes.

💡 Just like `Array.prototype.reduce()`

💡 If you need the current accumulated value on each emission, try [scan](#)!



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Sum a stream of numbers

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { reduce } from 'rxjs/operators';

const source = of(1, 2, 3, 4);
const example = source.pipe(reduce((acc, val) => acc + val));
//output: Sum: 10'
const subscribe = example.subscribe(val => console.log('Sum:', val));
```

Additional Resources

- [reduce](#) 📄 - Official docs
- [Scan\(\) vs reduce\(\) | RxJS TUTORIAL](#) 📺 - Academind
- [Build your own reduce operator](#) 📺 - Kwinten Pisman

 Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/reduce.ts>

scan

signature: `scan(accumulator: function, seed: any): Observable`

Reduce over time.


💡 You can create [Redux](#)-like state management with scan!

 Ultimate Courses™

**Fast-track your [RxJS skills](#) to build
[any project](#) or [get hired](#) at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Sum over time

([StackBlitz](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { scan } from 'rxjs/operators';

const source = of(1, 2, 3);
// basic scan example, sum over time starting with zero
const example = source.pipe(scan((acc, curr) => acc + curr, 0));
// log accumulated values
// output: 1,3,6
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Accumulating an object

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { Subject } from 'rxjs';
import { scan } from 'rxjs/operators';

const subject = new Subject();
//scan example building an object over time
const example = subject.pipe(
  scan((acc, curr) => Object.assign({}, acc, curr), {})
);
//log accumulated values
const subscribe = example.subscribe(val =>
  console.log('Accumulated object:', val)
);
//next values into subject, adding properties to object
// {name: 'Joe'}
subject.next({ name: 'Joe' });
// {name: 'Joe', age: 30}
subject.next({ age: 30 });
// {name: 'Joe', age: 30, favoriteLanguage: 'JavaScript'}
subject.next({ favoriteLanguage: 'JavaScript' });
```

Example 3: Emitting random values from the accumulated array.

([StackBlitz](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { scan, map, distinctUntilChanged } from 'rxjs/operators';

// Accumulate values in an array, emit random values from this array.
const scanObs = interval(1000)
  .pipe(
    scan((a, c) => [...a, c], []),
    map(r => r[Math.floor(Math.random() * r.length)]),
    distinctUntilChanged()
  )
  .subscribe(console.log);
```

Example 4: Accumulating http responses over time

([StackBlitz](#))

```
// RxJS v6+
import { interval, of } from 'rxjs';
import { scan, delay, repeat, mergeMap } from 'rxjs/operators';

const fakeRequest = of('response').pipe(delay(2000));








// output:
// ['response'],
// ['response', 'response'],
// ['response', 'response', 'response'],
// etc...

interval(1000)
  .pipe(
    mergeMap(_ => fakeRequest),
    scan<string>((all, current) => [...all, current], [])
  )
  .subscribe(console.log);
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Catch The Dot Game](#)
- [Click Ninja Game](#)
- [Flappy Bird Game](#)
- [Matrix Digital Rain](#)
- [Memory Game](#)
- [Platform Jumper Game](#)
- [Progress Bar](#)
- [Smart Counter](#)
- [Space Invaders Game](#)
- [Stop Watch](#)
- [Tank Battle Game](#)
- [Tetris Game](#)

Additional Resources

- [scan](#)  - Official docs
- [Aggregating streams with reduce and scan using RxJS](#)  - Ben Lesh
- [Updating data with scan](#)   - John Linquist
- [Transformation operator: scan](#)   - André Staltz
- [Build your own scan operator](#)  - Kwinten Pisman



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/scan.ts>

switchMap

signature: `switchMap(project: function: Observable, resultSelector: function(outerValue, innerValue, outerIndex, innerIndex): any): Observable`

Map to observable, complete previous inner observable, emit values.

💡 If you would like more than one inner subscription to be maintained, try `mergeMap` !

💡 This operator is generally considered a safer default to `mergeMap` !


💡 This operator can cancel in-flight network requests!

Why use `switchMap` ?

The main difference between `switchMap` and other flattening operators is the cancelling effect. On each emission the previous inner observable (the result of the function you supplied) is cancelled and the new observable is subscribed. You can remember this by the phrase **switch to a new observable**.

This works perfectly for scenarios like `typeaheads` where you are no longer concerned with the response of the previous request when a new input arrives. This also is a safe option in situations where a long lived inner observable could cause memory leaks, for instance if you used `mergeMap` with an interval and forgot to properly dispose of inner subscriptions. Remember, `switchMap` maintains only one inner subscription at a time, this can be seen clearly in the [first example](#).


Be careful though, you probably want to avoid `switchMap` in scenarios where every request needs to complete, think writes to a database. `switchMap` could cancel a request if the source emits quickly enough. In these scenarios [mergeMap](#) is the correct option.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Restart interval on every click

([StackBlitz](#))

```
// RxJS v6+
import { interval, fromEvent } from 'rxjs';
import { switchMap } from 'rxjs/operators';

fromEvent(document, 'click')
  .pipe(
    // restart counter on every click
    switchMap(() => interval(1000))
  )
  .subscribe(console.log);
```

Example 2: Countdown timer with pause and resume

([StackBlitz](#))

```
// RxJS v6+
import { interval, fromEvent, merge, empty } from 'rxjs';
import { switchMap, scan, takeWhile, startWith, mapTo } from 'rxjs/operators';

const COUNTDOWN_SECONDS = 10;

// elem refs
const remainingLabel = document.getElementById('remaining');
const pauseButton = document.getElementById('pause');
const resumeButton = document.getElementById('resume');

// streams
const interval$ = interval(1000).pipe(mapTo(-1));
const pause$ = fromEvent(pauseButton, 'click').pipe(mapTo(false));
const resume$ = fromEvent(resumeButton, 'click').pipe(mapTo(true));

const timer$ = merge(pause$, resume$)
  .pipe(
    startWith(true),
    switchMap(val => (val ? interval$ : empty())),
    scan((acc, curr) => (curr ? curr + acc : acc), COUNTDOWN_SECONDS),
    takeWhile(v => v >= 0)
  )
  .subscribe((val: any) => (remainingLabel.innerHTML = val));
```

Example 3: Using a `resultSelector` function

([StackBlitz](#))

```
// RxJS v6+
import { timer, interval } from 'rxjs';
import { switchMap } from 'rxjs/operators';

// switch to new inner observable when source emits, emit result
// of project function
timer(0, 5000)
  .pipe(
    switchMap(
      _ => interval(2000),
      (outerValue, innerValue, outerIndex, innerIndex) => ({
        outerValue,
        innerValue,
        outerIndex,
        innerIndex
      })
    )
  )
  /*
  Output:
  {outerValue: 0, innerValue: 0, outerIndex: 0, innerIndex: 0}
  {outerValue: 0, innerValue: 1, outerIndex: 0, innerIndex: 1}
  {outerValue: 1, innerValue: 0, outerIndex: 1, innerIndex: 0}
  {outerValue: 1, innerValue: 1, outerIndex: 1, innerIndex: 1}
  */
  .subscribe(console.log);
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Car Racing Game](#)
- [Catch The Dot Game](#)
- [HTTP Polling](#)
- [Lockscreen](#)
- [Memory Game](#)
- [Mine Sweeper Game](#)
- [Platform Jumper Game](#)

- [Progress Bar](#)
- [Smart Counter](#)
- [Stop Watch](#)
- [Typeahead](#)

Additional Resources

- [switchMap](#) 📄 - Official docs
- [Avoiding switchMap-Related Bugs](#) - Nicholas Jamieson
- [Starting a stream with switchMap](#) 🖥️ 💰 - John Linquist
- [Use RxJS switchMap to map and flatten higher order observables](#) 🖥️ 💰 - André Staltz
- [Use switchMap as a safe default to flatten observables in RxJS](#) 🖥️ 💰 - André Staltz
- [Build your own switchMap operator](#) 🖥️ - Kwinten Pisman



Source Code:

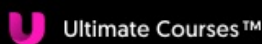
<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/switchMap.ts>

switchMapTo

signature: `switchMapTo(innerObservable: Observable, resultSelector: function(outerValue, innerValue, outerIndex, innerIndex): any): Observable`

Map to same inner observable, complete previous inner observable.

💡 If you need to consider the emitted value from the source, try `switchMap` !



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Restart countdown on click, until countdown completes one time

([StackBlitz](#))

```
// RxJS v6+
import { interval, fromEvent } from 'rxjs';
import {
  switchMapTo,
  scan,
  startWith,
  takeWhile,
  finalize
} from 'rxjs/operators';

const COUNTDOWN_TIME = 10;

// reference
const countdownElem = document.getElementById('countdown');

// streams
const click$ = fromEvent(document, 'click');
const countdown$ = interval(1000).pipe(
  scan((acc, _) => --acc, COUNTDOWN_TIME),
  startWith(COUNTDOWN_TIME)
);

click$
  .pipe(
    switchMapTo(countdown$),
    takeWhile(val => val >= 0),
    finalize(() => (countdownElem.innerHTML = "We're done here!"))
  )
  .subscribe((val: any) => (countdownElem.innerHTML = val));
```

Additional Resources

- [switchMapTo](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/switchMapTo.ts>

toArray

signature: `toArray(): OperatorFunction`


Collects all source emissions and emits them as an array when the source completes.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: get values emitted by interval as an array when interval completes

([StackBlitz](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { toArray, take } from 'rxjs/operators';

interval(100)
  .pipe(
    take(10),
    toArray()
  )
  .subscribe(console.log);

// output: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Related Recipes

- [Breakout Game](#)
- [Lockscreen](#)

Additional Resources

- [toArray](#)  - Official docs



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/toArray.ts>

window

signature: `window(windowBoundaries: Observable): Observable`

Observable of values for window of time.

Examples




Example 1: Open window specified by inner observable

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { timer, interval } from 'rxjs';
import { window, scan, mergeAll } from 'rxjs/operators';

//emit immediately then every 1s
const source = timer(0, 1000);
const example = source.pipe(window(interval(3000)));
const count = example.pipe(scan((acc, curr) => acc + 1, 0));
/*
  "Window 1:"
  0
  1
  2
  "Window 2:"
  3
  4
  5
  ...
*/
const subscribe = count.subscribe(val => console.log(`Window ${val}:`));
const subscribeTwo = example
  .pipe(mergeAll())
  .subscribe(val => console.log(val));
```

Additional Resources

- [window](#)  - Official docs
 - [Split an RxJS observable with window](#)   - André Staltz
-




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/window.ts>

windowCount

signature: `windowCount(windowSize: number, startWindowEvery: number): Observable`


Observable of values from source, emitted each time provided count is fulfilled.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone

Creator of LearnRxJS.io

Examples

Example 1: Start new window every x items emitted

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval } from 'rxjs';
import { windowCount, mergeAll, tap } from 'rxjs/operators';

//emit every 1s
const source = interval(1000);
const example = source.pipe(
  //start new window every 4 emitted values
  windowCount(4),
  tap(_ => console.log('NEW WINDOW!'))
);

const subscribeTwo = example
  .pipe(
    //window emits nested observable
    mergeAll()
    /*
      output:
      "NEW WINDOW!"
      0
      1
      2
      3
      "NEW WINDOW!"
      4
      5
      6
      7
    */
  )
  .subscribe(val => console.log(val));
```

Additional Resources

- [windowCount](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/windowCount.ts>

windowTime

signature: `windowTime(windowTimeSpan: number, windowCreationInterval: number, scheduler: Scheduler): Observable`


Observable of values collected from source for each provided time span.

 Ultimate Courses™

Fast-track your **RxJS skills to build any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Open new window every specified duration

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))


```
// RxJS v6+
import { timer } from 'rxjs';
import { windowTime, tap, mergeAll } from 'rxjs/operators';

//emit immediately then every 1s
const source = timer(0, 1000);
const example = source.pipe(
  //start new window every 3s
  windowTime(3000),
  tap(_ => console.log('NEW WINDOW!'))
);

const subscribeTwo = example
  .pipe(
    //window emits nested observable
    mergeAll()
    /*
      output:
      "NEW WINDOW!"
      0
      1
      2
      "NEW WINDOW!"
      3
      4
      5
    */
  )
  .subscribe(val => console.log(val));
```

Additional Resources

- [windowTime](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/windowTime.ts>

windowToggle

signature: `windowToggle(openings: Observable,
closingSelector: function(value): Observable): Observable`


Collect and emit observable of values from source between opening and closing emission.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Toggle window at increasing interval


([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { timer, interval } from 'rxjs';
import { tap, windowToggle, mergeAll } from 'rxjs/operators';

//emit immediately then every 1s
const source = timer(0, 1000);
//toggle window on every 5
const toggle = interval(5000);
const example = source.pipe(
  //turn window on every 5s
  windowToggle(toggle, val => interval(val * 1000)),
  tap(_ => console.log('NEW WINDOW!'))
);

const subscribeTwo = example
  .pipe(
    //window emits nested observable
    mergeAll()
    /*
      output:
      "NEW WINDOW!"
      5
      "NEW WINDOW!"
      10
      11
      "NEW WINDOW!"
      15
      16
      "NEW WINDOW!"
      20
      21
      22
    */
  )
  .subscribe(val => console.log(val));
```

Additional Resources

- [windowToggle](#)  - Official docs
- [Split an RxJS observable conditionally with windowToggle](#)   - André

Staltz




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/windowToggle.ts>

windowWhen

signature: `windowWhen(closingSelector: function(): Observable): Observable`


Close window at provided time frame emitting observable of collected values from source.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone

Creator of LearnRxJS.io

Examples

Example 1: Open and close window at interval

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, timer } from 'rxjs';
import { windowWhen, tap, mergeAll } from 'rxjs/operators';

//emit immediately then every 1s
const source = timer(0, 1000);
const example = source.pipe(
  //close window every 5s and emit observable of collected values from source
  windowWhen(() => interval(5000)),
  tap(_ => console.log('NEW WINDOW!'))
);

const subscribeTwo = example
  .pipe(
    //window emits nested observable
    mergeAll()
  )
  /*
    output:
    "NEW WINDOW!"
    0
    1
    2
    3
    4
    "NEW WINDOW!"
    5
    6
    7
    8
    9
  */
)
.subscribe(val => console.log(val));
```

Additional Resources

- [windowWhen](#)  - Official docs

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/windowWhen.ts>

Utility Operators

From logging, handling notifications, to setting up schedulers, these operators provide helpful utilities in your observable toolkit.

Contents

- [tap / do](#) ★
- [delay](#) ★
- [delayWhen](#)
- [dematerialize](#)
- [finalize / finally](#)
- [let](#)
- [repeat](#)
- [repeatWhen](#)
- [timeInterval](#)
- [timeout](#)
- [timeoutWith](#)
- [toPromise](#)

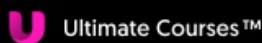
★ - *commonly used*

tap / do

signature: `tap(nextOrObserver: function, error: function, complete: function): Observable`

Transparently perform actions or side-effects, such as logging.

💡 If you are using as a pipeable operator, `do` is known as `tap` !



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Logging with tap

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { tap, map } from 'rxjs/operators';

const source = of(1, 2, 3, 4, 5);
// transparently log values from source with 'tap'
const example = source.pipe(
  tap(val => console.log(`BEFORE MAP: ${val}`)),
  map(val => val + 10),
  tap(val => console.log(`AFTER MAP: ${val}`))
);

// 'tap' does not transform values
// output: 11...12...13...14...15
const subscribe = example.subscribe(val => console.log(val));
```

Example 2: Using tap with object

([StackBlitz](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { tap, map } from 'rxjs/operators';

const source = of(1, 2, 3, 4, 5);

// tap also accepts an object map to log next, error, and complete
const example = source
  .pipe(
    map(val => val + 10),
    tap({
      next: val => {
        // on next 11, etc.
        console.log('on next', val);
      },
      error: error => {
        console.log('on error', error.message);
      },
      complete: () => console.log('on complete')
    })
  )
// output: 11, 12, 13, 14, 15
.subscribe(val => console.log(val));
```

Related Recipes

- [Battleship Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Catch The Dot Game](#)
- [Click Ninja Game](#)
- [Flappy Bird Game](#)
- [Horizontal Scroll Indicator](#)
- [Lockscreen](#)
- [Memory Game](#)
- [Mine Sweeper Game](#)
- [Platform Jumper Game](#)

- [Save Indicator](#)
- [Space Invaders Game](#)
- [Stop Watch](#)
- [Swipe To Refresh](#)
- [Tank Battle Game](#)
- [Tetris Game](#)
- [Type Ahead](#)

Additional Resources

- [tap](#) 📄 - Official docs
- [Logging a stream with do](#) 🖨️ 💻 - John Linquist
- [Utility operator: do](#) 🖨️ 💻 - André Staltz
- [Build your own tap operator](#) 🖨️ - Kwinten Pisman




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/do.ts>

delay

signature: `delay(delay: number | Date, scheduler: Scheduler): Observable`


Delay emitted values by given time.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Delay to recognize long press

([StackBlitz](#))

```
import { fromEvent, of } from 'rxjs';
import { mergeMap, delay, takeUntil } from 'rxjs/operators';

const mousedown$ = fromEvent(document, 'mousedown');
const mouseup$ = fromEvent(document, 'mouseup');

mousedown$
  .pipe(
    mergeMap(event =>
      of(event).pipe(
        delay(700),
        takeUntil(mouseup$)
      )
    )
  )
  .subscribe(event => console.log('Long Press!', event));
```

Example 2: Delay for increasing durations

([StackBlitz](#))

```
// RxJS v6+
import { of, merge } from 'rxjs';
import { mapTo, delay } from 'rxjs/operators';

//emit one item
const example = of(null);
//delay output of each by an extra second
const message = merge(
  example.pipe(mapTo('Hello')),
  example.pipe(
    mapTo('World!'),
    delay(1000)
  ),
  example.pipe(
    mapTo('Goodbye'),
    delay(2000)
  ),
  example.pipe(
    mapTo('World!'),
    delay(3000)
  )
);
//output: 'Hello'...'World!'...'Goodbye'...'World!'
const subscribe = message.subscribe(val => console.log(val));
```

Related Recipes

- [Battleship Game](#)
- [Progress Bar](#)
- [Save Indicator](#)
- [Swipe To Refresh](#)

Additional Resources

- [delay](#)  - Official docs
- [Transformation operator: delay and delayWhen](#)   - André Staltz




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/delay.ts>

delayWhen

signature: `delayWhen(selector: Function, sequence: Observable): Observable`


Delay emitted values determined by provided function.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Delay based on observable

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { interval, timer } from 'rxjs';
import { delayWhen } from 'rxjs/operators';

//emit value every second
const message = interval(1000);
//emit value after five seconds
const delayForFiveSeconds = () => timer(5000);
//after 5 seconds, start emitting delayed interval values
const delayWhenExample = message.pipe(delayWhen(delayForFiveSeconds));
//log values, delayed for 5 seconds
//ex. output: 5s....1...2...3
const subscribe = delayWhenExample.subscribe(val => console.log(val));
```

Additional Resources

- [delayWhen](#) 📖 - Official docs
- [Transformation operator: delay and delayWhen](#) 🖥️ 📄 - André Staltz


📁 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/delayWhen.ts>

dematerialize

signature: `dematerialize(): Observable`


Turn notification objects into notification values.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: Converting notifications to values

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { from, Notification } from 'rxjs';
import { dematerialize } from 'rxjs/operators';

//emit next and error notifications
const source = from([
  Notification.createNext('SUCCESS!'),
  Notification.createError('ERROR!')
]).pipe(
  //turn notification objects into notification values
  dematerialize()
);

//output: 'NEXT VALUE: SUCCESS' 'ERROR VALUE: 'ERROR!'
const subscription = source.subscribe({
  next: val => console.log(`NEXT VALUE: ${val}`),
  error: val => console.log(`ERROR VALUE: ${val}`)
});
```

Additional Resources

- [dematerialize](#)  - Official docs

 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/dematerialize.ts>

finalize / finally

signature: `finalize(callback: () => void)`

Call a function when observable completes or errors

Examples

Example 1: Execute callback function when the observable completes

([StackBlitz](#))

```
import { interval } from 'rxjs';
import { take, finalize } from 'rxjs/operators';

//emit value in sequence every 1 second
const source = interval(1000);
//output: 0,1,2,3,4,5....
const example = source.pipe(
  take(5), //take only the first 5 values
  finalize(() => console.log('Sequence complete')) // Execute wh
  en the observable completes
)
const subscribe = example.subscribe(val => console.log(val));
```

Related Recipes

- [Battleship Game](#)
- [Car Racing Game](#)
- [Click Ninja Game](#)
- [HTTP Polling](#)
- [Mine Sweeper Game](#)
- [Swipe To Refresh](#)
- [Tetris Game](#)

Additional Resources

- [finalize](#)  - Official docs



Source Code:

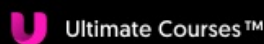
<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/finalize.ts>

let

signature: `let(function): Observable`

Let me have the whole observable.

⚠ `let` is no longer available, or necessary, with pipeable operators! (RxJS 5.5+)



**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: Reusing error handling logic with let

([jsBin](#) | [jsFiddle](#))


```
// custom error handling logic
const retryThreeTimes = obs =>
  obs.retry(3).catch(_ => Rx.Observable.of('ERROR!'));
const examplePromise = val =>
  new Promise(resolve => resolve(`Complete: ${val}`));

//faking request
const subscribe = Rx.Observable.of('some_url')
  .mergeMap(url => examplePromise(url))
  // could reuse error handling logic in multiple places with let

  .let(retryThreeTimes)
  //output: Complete: some_url
  .subscribe(result => console.log(result));

const customizableRetry = retryTimes => obs =>
  obs.retry(retryTimes).catch(_ => Rx.Observable.of('ERROR!'));

//faking request
const secondSubscribe = Rx.Observable.of('some_url')
  .mergeMap(url => examplePromise(url))
  // could reuse error handling logic in multiple places with let

  .let(customizableRetry(3))
  //output: Complete: some_url
  .subscribe(result => console.log(result));
```

Example 2: Applying map with let

([jsBin](#) | [jsFiddle](#))

```
//emit array as a sequence
const source = Rx.Observable.from([1, 2, 3, 4, 5]);
//demonstrating the difference between let and other operators
const test = source
  .map(val => val + 1)
  /*
    this would fail, let behaves differently than most operators
    val in this case is an observable
  */
  //.let(val => val + 2)
  .subscribe(val => console.log('VALUE FROM ARRAY: ', val));

const subscribe = source
  .map(val => val + 1)
  //'let' me have the entire observable
  .let(obs => obs.map(val => val + 2))
  //output: 4,5,6,7,8
  .subscribe(val => console.log('VALUE FROM ARRAY WITH let: ', val));
```

Example 3: Applying multiple operators with let

([jsBin](#) | [jsFiddle](#))

```
//emit array as a sequence
const source = Rx.Observable.from([1, 2, 3, 4, 5]);

//let provides flexibility to add multiple operators to source observable then return
const subscribeTwo = source
  .map(val => val + 1)
  .let(obs =>
    obs
      .map(val => val + 2)
      //also, just return evens
      .filter(val => val % 2 === 0)
  )
  //output: 4,6,8
  .subscribe(val => console.log('let WITH MULTIPLE OPERATORS: ', val));
```

Example 4: Applying operators through function

([jsBin](#) | [jsFiddle](#))

```
//emit array as a sequence
const source = Rx.Observable.from([1, 2, 3, 4, 5]);

//pass in your own function to add operators to observable
const obsArrayPlusYourOperators = yourAppliedOperators => {
  return source.map(val => val + 1).let(yourAppliedOperators);
};
const addTenThenTwenty = obs => obs.map(val => val + 10).map(val => val + 20);
const subscribe = obsArrayPlusYourOperators(addTenThenTwenty)
  //output: 32, 33, 34, 35, 36
  .subscribe(val => console.log('let FROM FUNCTION:', val));
```

Additional Resources

- [let](#)  - Official docs



Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/let.ts>

repeat

signature: `repeat(count: number): Observable`

Repeats an observable on completion.


💡 Like `retry` but for non error cases!

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Repeat 3 times

([StackBlitz](#))

```
// RxJS v6+
import { repeat, delay } from 'rxjs/operators';
import { of } from 'rxjs';


const delayedThing = of('delayed value').pipe(delay(2000));

delayedThing
  .pipe(repeat(3))
  // delayed value...delayed value...delayed value
  .subscribe(console.log);
```

Related Recipes

- [Click Ninja Game](#)
- [Lockscreen](#)
- [Space Invaders Game](#)
- [Swipe To Refresh](#)

Additional Resources

- [repeat](#)  - Official docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/repeat.ts>

timeInterval

signature: `timeInterval(scheduler: *):
Observable<TimeInterval<any>> | WebSocketSubject<T> |
Observable<T>`


Convert an Observable that emits items into one that emits indications of the amount of time elapsed between those emissions

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Time between mouse clicks

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { timeInterval, tap } from 'rxjs/operators';

fromEvent(document, 'mousedown')
  .pipe(
    timeInterval(),
    tap(console.log)
  )
  .subscribe(
    i =>
      (document.body.innerText = `milliseconds since last click:
${i.interval}`)
  );
```

Related Recipes

- [Click Ninja Game](#)

Additional Resources

- [timeInterval](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/timeInterval.ts>

timeout

signature: `timeout(due: number, scheduler: Scheduler): Observable`


Error if no value is emitted before specified duration

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Timeout after 2.5 seconds

([StackBlitz](#) | [jsBin](#) | [jsFiddle](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { concatMap, timeout, catchError, delay } from 'rxjs/operators';

// simulate request
function makeRequest(timeToDelay) {
  return of('Request Complete!').pipe(delay(timeToDelay));
}

of(4000, 3000, 2000)
  .pipe(
    concatMap(duration =>
      makeRequest(duration).pipe(
        timeout(2500),
        catchError(error => of(`Request timed out after: ${duration}`))
      )
    )
  )
  /*
  * "Request timed out after: 4000"
  * "Request timed out after: 3000"
  * "Request Complete!"
  */
  .subscribe(val => console.log(val));
```

Additional Resources

- [timeout](#)  - Official Docs


 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/timeout.ts>

timeoutWith

signature: `timeoutWith(due: number | Date, withObservable: ObservableInput, scheduler: SchedulerLike = async): OperatorFunction`


Subscribe to second Observable if no emission occurs in given time span.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](https://learnrxjs.io)

Examples

Example 1: Timeout after 1 second

([StackBlitz](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { timeoutWith, delay, concatMap } from 'rxjs/operators';

const fakeRequest = delayTime => of('!response!').pipe(delay(delayTime));
const requestTimeoutLogger = of('logging request timeout');
const timeoutThreshold = 1000;

of(timeoutThreshold + 1, timeoutThreshold - 1, timeoutThreshold + 3)
  .pipe(
    concatMap(e =>
      fakeRequest(e).pipe(timeoutWith(timeoutThreshold, requestTimeoutLogger))
    )
  )
  .subscribe(console.log);

/*
  OUTPUT:
    logging request timeout
    !response!
    logging request timeout
*/
```

Additional Resources

- [timeoutWith](#)  - Official Docs

 Source Code:


<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/timeoutWith.ts>

toPromise

signature: `toPromise() : Promise`

Convert observable to promise.


⚠️ `toPromise` is not a pipable operator, as it does not return an observable.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: Basic Promise

([jsBin](#) | [jsFiddle](#))

```
//return basic observable
const sample = val => Rx.Observable.of(val).delay(5000);
//convert basic observable to promise
const example = sample('First Example')
  .toPromise()
  //output: 'First Example'
  .then(result => {
    console.log('From Promise:', result);
  });
```

Example 2: Using Promise.all

([jsBin](#) | [jsFiddle](#))

```
//return basic observable
const sample = val => Rx.Observable.of(val).delay(5000);
/*
  convert each to promise and use Promise.all
  to wait for all to resolve
*/
const example = () => {
  return Promise.all([
    sample('Promise 1').toPromise(),
    sample('Promise 2').toPromise()
  ]);
};
//output: ["Promise 1", "Promise 2"]
example().then(val => {
  console.log('Promise.all Result:', val);
});
```

Additional Resources

- [toPromise](#)  - Official Docs



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/operators/toPromise.ts>

RxJS Operators By Example

A complete list of RxJS operators with clear explanations, relevant resources, and executable examples.

Prefer a split by operator type?

Contents (In Alphabetical Order)




- [ajax](#) ★
- [audit](#)
- [auditTime](#)
- [buffer](#)
- [bufferCount](#)
- [bufferTime](#) ★
- [bufferToggle](#)
- [bufferWhen](#)
- [catch / catchError](#) ★
- [combineAll](#)
- [combineLatest](#) ★
- [concat](#) ★
- [concatAll](#)
- [concatMap](#) ★
- [concatMapTo](#)
- [create](#)
- [debounce](#)
- [debounceTime](#) ★
- [defaultIfEmpty](#)
- [defer](#)
- [delay](#)
- [delayWhen](#)
- [distinct](#)
- [distinctUntilChanged](#) ★
- [distinctUntilKeyChanged](#)
- [endWith](#)

- `tap / do` ★
- `empty`
- `every`
- `exhaustMap`
- `expand`
- `filter` ★
- `finalize / finally`
- `find`
- `first`
- `forkJoin`
- `from` ★
- `fromEvent`
- `generate`
- `groupBy`
- `iif`
- `ignoreElements`
- `interval`
- `last`
- `let`
- `map` ★
- `mapTo`
- `merge` ★
- `mergeAll`
- `mergeMap / flatMap` ★
- `mergeScan`
- `multicast`
- `of` ★
- `partition`
- `pluck`
- `publish`
- `race`
- `range`
- `repeat`
- `repeatWhen`
- `retry`
- `retryWhen`

- [sample](#)
- [scan](#) ★
- [sequenceEqual](#)
- [share](#) ★
- [shareReplay](#) ★
- [single](#)
- [skip](#)
- [skipUntil](#)
- [skipWhile](#)
- [startWith](#) ★
- [switchMap](#) ★
- [switchMapTo](#)
- [take](#) ★
- [takeLast](#)
- [takeUntil](#) ★
- [takeWhile](#)
- [throttle](#)
- [throttleTime](#)
- [throw](#)
- [timeInterval](#)
- [timeout](#)
- [timeoutWith](#)
- [timer](#)
- [toArray](#)
- [toPromise](#)
- [window](#)
- [windowCount](#)
- [windowTime](#)
- [windowToggle](#)
- [windowWhen](#)
- [withLatestFrom](#) ★
- [zip](#)

★ - *commonly used*

Additional Resources

- [What Are Operators?](#)  - Official Docs
- [What Operators Are](#)   - André Staltz

Subjects

A Subject is a special type of Observable which shares a single execution path among observers.

You can think of this as a single speaker talking at a microphone in a room full of people. Their message (the subject) is being delivered to many (multicast) people (the observers) at once. This is the basis of **multicasting**. Typical observables would be comparable to a 1 on 1 conversation.

There are 4 variants of subjects:

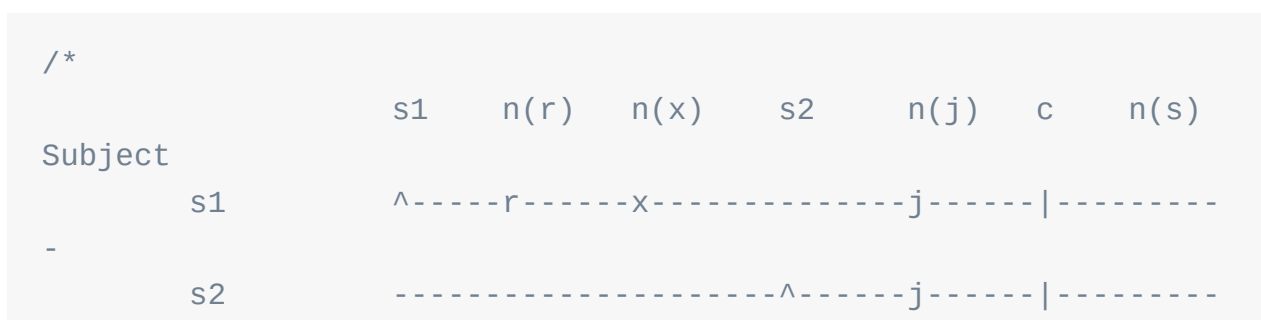
- **Subject** - No initial value or replay behavior.
- **AsyncSubject** - Emits latest value to observers upon completion.
- **BehaviorSubject** - Requires an initial value and emits its current value (last emitted item) to new subscribers.
- **ReplaySubject** - Emits specified number of last emitted values (a replay) to new subscribers.

Contents

- [AsyncSubject](#)
- [BehaviorSubject](#)
- [ReplaySubject](#)
- [Subject](#)

Subjects comparison

([Stackblitz](#))



```

-
AsyncSubject
    s1      ^-----j|-----
-
    s2      -----^-----j|-----
-
BehaviorSubject
    s1      ^a----r-----x-----j-----|-----
-
    s2      -----^x----j-----|-----
-
ReplaySubject
    s1      ^-----r-----x-----j-----|-----
-
    s2      -----^r-x---j-----|-----
-
*/

// RxJS v6+
import { Subject, AsyncSubject, BehaviorSubject, ReplaySubject }
from 'rxjs';

const subject = new Subject();
const asyncSubject = new AsyncSubject();
const behaviorSubject = new BehaviorSubject('a');
const replaySubject = new ReplaySubject(2);

const subjects = [subject, asyncSubject, behaviorSubject, replaySubject];
const log = subjectType => e => console.log(`${subjectType}: ${e}`);

console.log('SUBSCRIBE 1');
subject.subscribe(log('s1 subject'));
asyncSubject.subscribe(log('s1 asyncSubject'));
behaviorSubject.subscribe(log('s1 behaviorSubject'));
replaySubject.subscribe(log('s1 replaySubject'));

console.log('\nNEXT(r)');
subjects.forEach(o => o.next('r'));

console.log('\nNEXT(x)');

```

```
subjects.forEach(o => o.next('x'));

console.log('\nSUBSCRIBE 2');
subject.subscribe(log('s2 subject'));
asyncSubject.subscribe(log('s2 asyncSubject'));
behaviorSubject.subscribe(log('s2 behaviorSubject'));
replaySubject.subscribe(log('s2 replaySubject'));

console.log('\nNEXT(j)');
subjects.forEach(o => o.next('j'));

console.log('\nCOMPLETE');
subjects.forEach(o => o.complete());

console.log('\nNEXT(s)');
subjects.forEach(o => o.next('s'));

/*
OUTPUT:

SUBSCRIBE 1
s1 behaviorSubject: a

NEXT(r)
s1 subject: r
s1 behaviorSubject: r
s1 replaySubject: r

NEXT(x)
s1 subject: x
s1 behaviorSubject: x
s1 replaySubject: x

SUBSCRIBE 2
s2 behaviorSubject: x
s2 replaySubject: r
s2 replaySubject: x

NEXT(j)
s1 subject: j
s2 subject: j
s1 behaviorSubject: j
```

```
s2 behaviorSubject: j  
s1 replaySubject: j  
s2 replaySubject: j
```



COMPLETE

```
s1 asyncSubject: j  
s2 asyncSubject: j
```

NEXT(s)


```
* /
```

Additional Resources

- [Official Overview](#) 
- [Official Documentation](#) 
- [On The Subject Of Subjects \(in RxJS\)](#)  - Ben Lesh

AsyncSubject


Emits its last value on completion

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: simple AsyncSubject

([Stackblitz](#))

```
// RxJS v6+
import { AsyncSubject } from 'rxjs';

const sub = new AsyncSubject();

sub.subscribe(console.log);

sub.next(123); //nothing logged

sub.subscribe(console.log);

sub.next(456); //nothing logged
sub.complete(); //456, 456 logged by both subscribers
```

Additional Resources

- [AsyncSubject](#)  - Official docs

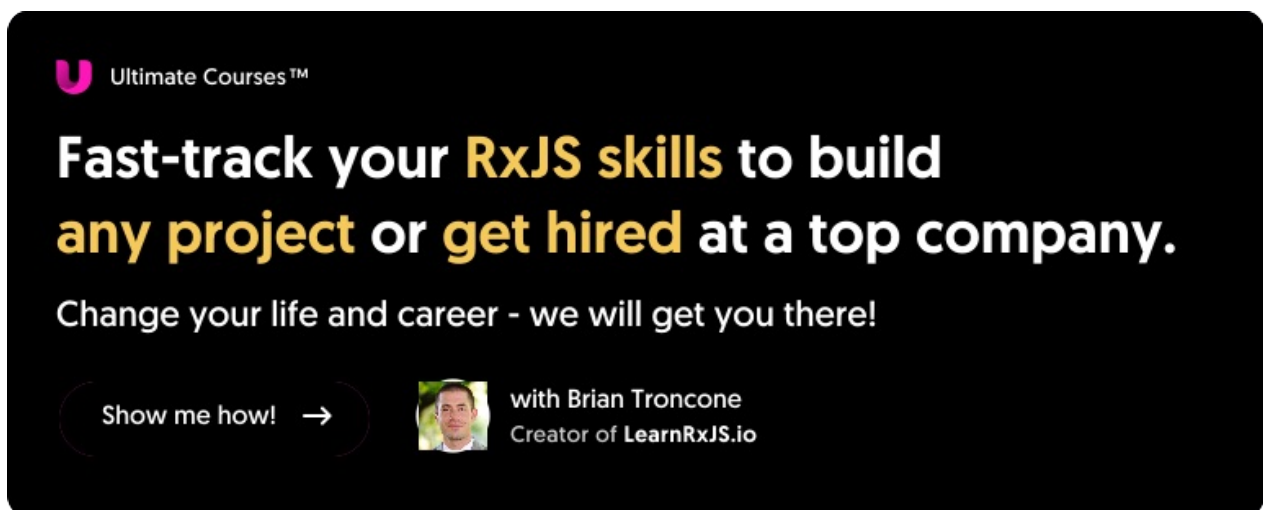
 Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/AsyncSubject.ts>

BehaviorSubject

Requires an initial value and emits the current value to new subscribers

💡 If you want the last emitted value(s) on subscription, but do not need to supply a seed value, check out [ReplaySubject](#) instead!




U Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1: Simple BehaviorSubject

([Stackblitz](#))

```
// RxJS v6+
import { BehaviorSubject } from 'rxjs';

const subject = new BehaviorSubject(123);

// two new subscribers will get initial value => output: 123, 123

subject.subscribe(console.log);
subject.subscribe(console.log);

// two subscribers will get new value => output: 456, 456
subject.next(456);

// new subscriber will get latest value (456) => output: 456
subject.subscribe(console.log);

// all three subscribers will get new value => output: 789, 789, 789
subject.next(789);

// output: 123, 123, 456, 456, 456, 789, 789, 789
```

Example 2: BehaviorSubject with new subscribers created on mouse clicks

([Stackblitz](#))

```
// RxJS v6+
import { BehaviorSubject, fromEvent, interval, merge } from 'rxjs';
import { map, tap, mergeMap } from 'rxjs/operators';

const setElementText = (elemId, text) =>
  (document.getElementById(elemId).innerText = text.toString());
const addHtmlElement = coords =>
  (document.body.innerHTML += `
    <div
      id=${coords.id}
      style="
        position: absolute;
        height: 30px;
```

```
    width: 30px;
    text-align: center;
    top: ${coords.y}px;
    left: ${coords.x}px;
    background: silver;
    border-radius: 80%;"
  >
</div>`);
```

```
const subject = new BehaviorSubject(0);

const click$ = fromEvent(document, 'click').pipe(
  map((e: MouseEvent) => ({
    x: e.clientX,
    y: e.clientY,
    id: Math.random()
  })),
  tap(addHtmlElement),
  mergeMap(coords => subject.pipe(tap(v => setElementText(coords.
id, v))))
);

const interval$ = interval(1000).pipe(
  tap(v => subject.next(v)),
  tap(v => setElementText('intervalValue', v))
);

merge(click$, interval$).subscribe();
```

Related Recipes

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Car Racing Game](#)

Additional Resources

- [BehaviorSubject](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/BehaviorSubject.ts>

ReplaySubject


"Replays" or emits old values to new subscribers

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Examples

Example 1: simple ReplaySubject

([Stackblitz](#))

```
// RxJS v6+
import { ReplaySubject } from 'rxjs';

const sub = new ReplaySubject(3);

sub.next(1);
sub.next(2);
sub.subscribe(console.log); // OUTPUT => 1,2
sub.next(3); // OUTPUT => 3
sub.next(4); // OUTPUT => 4
sub.subscribe(console.log); // OUTPUT => 2,3,4 (log of last 3 values from new subscriber)
sub.next(5); // OUTPUT => 5,5 (log from both subscribers)
```

Additional Resources

- [ReplaySubject](#)  - Official docs




Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/ReplaySubject.ts>

Subject


A special type of Observable which shares a single execution path among observers

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Examples

Example 1: simple Subject

([Stackblitz](#))

```
// RxJS v6+
import { Subject } from 'rxjs';

const sub = new Subject();

sub.next(1);
sub.subscribe(console.log);
sub.next(2); // OUTPUT => 2
sub.subscribe(console.log);
sub.next(3); // OUTPUT => 3,3 (logged from both subscribers)
```

Related Recipes

- [Battleship Game](#)
- [Lockscreen](#)

Additional Resources

- [Subject](#)  - Official docs



Source Code:

<https://github.com/ReactiveX/rxjs/blob/master/src/internal/Subject.ts>

Recipes

Common use-cases and interesting recipes to help learn RxJS.


Contents

- [Alphabet Invasion Game](#)
- [Battleship Game](#)
- [Breakout Game](#)
- [Car Racing Game](#)
- [Catch The Dot Game](#)
- [Click Ninja Game](#)
- [Flappy Bird Game](#)
- [Game Loop](#)
- [Horizontal Scroll Indicator](#)
- [HTTP Polling](#)
- [Lockscreen](#)
- [Matrix Digital Rain](#)
- [Memory Game](#)
- [Mine Sweeper Game](#)
- [Platform Jumper Game](#)
- [Progress Bar](#)
- [Save Indicator](#)
- [Smart Counter](#)
- [Space Invaders Game](#)
- [Stop Watch](#)
- [Swipe To Refresh](#)
- [Tank Battle Game](#)
- [Tetris Game](#)
- [Type Ahead](#)

Alphabet Invasion Game

By [adamlubek](#)


This recipe demonstrates RxJS implementation of Alphabet Invasion Game.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))

Score: 0, Level: 1
u



index.ts

```
// RxJS v6+
import { interval, fromEvent, combineLatest, BehaviorSubject } from 'rxjs';
import { scan, startWith, map, takeWhile, switchMap } from 'rxjs/operators';
import { State, Letter, Letters } from './interfaces';

const randomLetter = () =>
  String.fromCharCode(
    Math.random() * ('z'.charCodeAt(0) - 'a'.charCodeAt(0)) + 'a'.charCodeAt(0)
  );

const levelChangeThreshold = 20;
const speedAdjust = 50;
const endThreshold = 15;
const gameWidth = 30;

const intervalSubject = new BehaviorSubject(600);

const letters$ = intervalSubject.pipe(
  switchMap(i =>
    interval(i).pipe(
      scan < number,
      Letters >
      (letters => ({
        intrvl: i,
        ltrs: [
          {
            letter: randomLetter(),
            yPos: Math.floor(Math.random() * gameWidth)
          },
          ...letters.ltrs
        ]
      })),
      { ltrs: [], intrvl: 0 })
    )
  );

const keys$ = fromEvent(document, 'keydown').pipe(
  startWith({ key: '' })
```

```

    map((e: KeyboardEvent) => e.key)
  );

  const renderGame = (state: State) => (
    (document.body.innerHTML = `Score: ${state.score}, Level: ${state.level} <br/>`),
    state.letters.forEach(
      l =>
        (document.body.innerHTML += '&nbsp;'.repeat(l.yPos) + l.letter + '<br/>')
    ),
    (document.body.innerHTML +=
      '<br/>'.repeat(endThreshold - state.letters.length - 1) +
      '-'.repeat(gameWidth))
  );

  const renderGameOver = () => (document.body.innerHTML += '<br/>GAME OVER!');
  const noop = () => {};

  const game$ = combineLatest(keys$, letters$).pipe(
    scan < [string, Letters, State] >
      ((state, [key, letters]) => (
        letters.ltrs[letters.ltrs.length - 1] &&
        letters.ltrs[letters.ltrs.length - 1].letter === key
          ? ((state.score = state.score + 1), letters.ltrs.pop())
          : noop,
        state.score > 0 && state.score % levelChangeThreshold === 0

          ? ((letters.ltrs = []),
            (state.level = state.level + 1),
            (state.score = state.score + 1),
            intervalSubject.next(letters.intrvl - speedAdjust))
          : noop,
        { score: state.score, letters: letters.ltrs, level: state.level }
      )),
    { score: 0, letters: [], level: 1 }
  ),
  takeWhile(state => state.letters.length < endThreshold)
);

game$.subscribe(renderGame, noop, renderGameOver);

```



interfaces.ts

```
export interface Letter {  
  letter: String;  
  yPos: number;  
}  
export interface Letters {  
  ltrs: Letter[];  
  intrvl: number;  
}  
export interface State {  
  score: number;  
  letters: Letter[];  
  level: number;  
}
```


Operators Used

- [BehaviorSubject](#)
- [combineLatest](#)
- [fromEvent](#)
- [interval](#)
- [map](#)
- [scan](#)
- [startWith](#)
- [switchMap](#)
- [takeWhile](#)

Battleship Game

By [adamlubek](#)


This recipe demonstrates an RxJS implementation of Battleship Game where you play against the computer.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

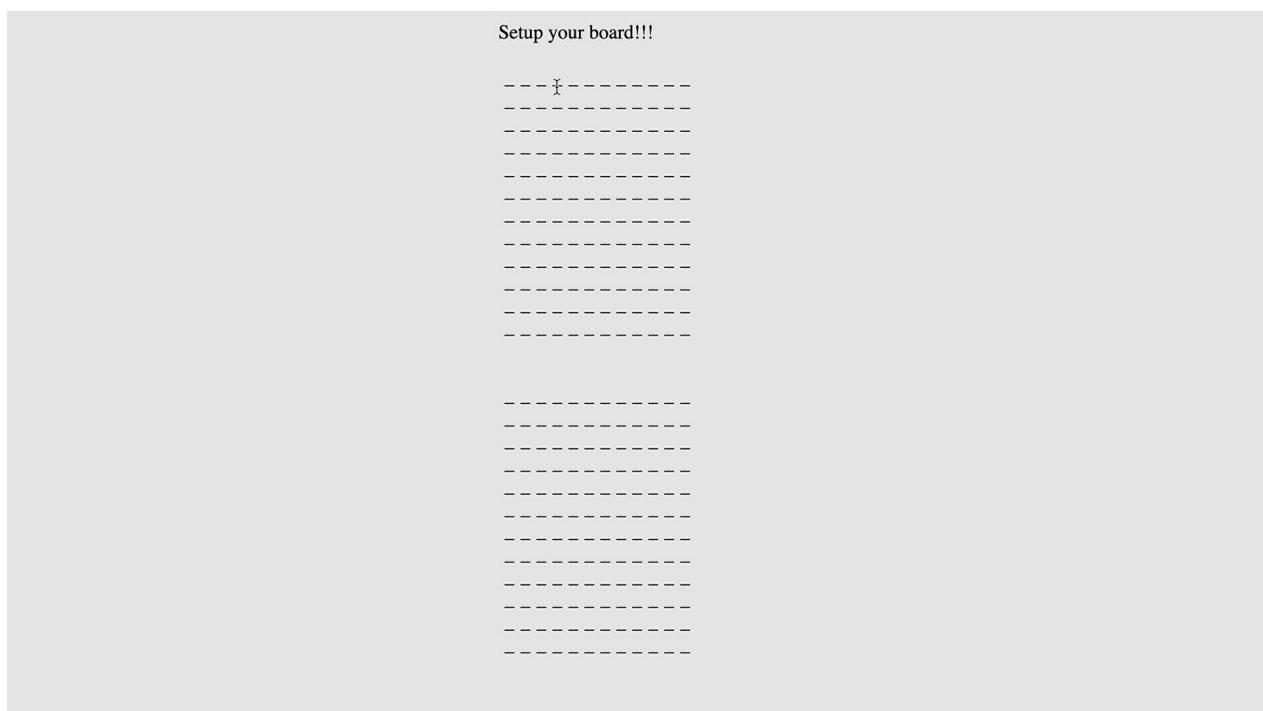
Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))



index.ts

```
// RxJS v6+
import { concat, merge } from 'rxjs';
import { switchMap, takeWhile, finalize } from 'rxjs/operators';
import { NUMBER_OF_SHIP_PARTS } from './constants';
import { displayGameOver, paintBoards$ } from './html-renderer';
import { shots$, computerScore$, playerScore$, isNotGameOver } from './game';
import { setup$, emptyBoards$ } from './setup';
import { Boards } from './interfaces';

const game$ = emptyBoards$.pipe(
  paintBoards$,
  switchMap((boards: Boards) =>
    concat(setup$(boards), shots$(boards)).pipe(
      takeWhile(isNotGameOver),
      finalize(displayGameOver(computerScore$))
    )
  )
);

merge(game$, computerScore$, playerScore$).subscribe();
```

setup.ts

```
import { concat, interval, of, fromEvent, pipe, noop } from 'rxjs';
import { filter, map, scan, take, tap } from 'rxjs/operators';
import {
  GAME_SIZE,
  NUMBER_OF_SHIP_PARTS,
  EMPTY,
  COMPUTER,
  PLAYER
} from './constants';
import {
  paintBoards$,
  computerScoreContainer,
  playerScoreContainer
```



```

} from './html-renderer';
import { random, validClicks$ } from './game';
import { Boards } from './interfaces';

const isThereEnoughSpaceForNextMove = (
  board: number[][],
  ship: number,
  x: number,
  y: number
) => {
  const row = [...board[x]];
  row[y] = ship;
  const col = board.map(r => r.filter((c, j) => j === y)[0]);
  col[x] = ship;

  const shipStartInCol = col.indexOf(ship);
  const shipEndInCol = col.lastIndexOf(ship);
  const shipStartInRow = row.indexOf(ship);
  const shipEndInRow = row.lastIndexOf(ship);

  const checkSpace = (arr, start, end) => {
    const startIndex = arr.lastIndexOf(
      (e, i) => e !== EMPTY && e !== ship && i < start
    );
    const endIndex = arr.findIndex(
      (e, i) => e !== EMPTY && e !== ship && i > end
    );
    const room = arr.slice(startIndex + 1, endIndex);
    return room.length >= ship;
  };

  return shipStartInCol !== shipEndInCol
    ? checkSpace(col, shipStartInCol, shipEndInCol)
    : shipStartInRow !== shipEndInRow
    ? checkSpace(row, shipStartInRow, shipEndInRow)
    : true;
};

const getTwoValidMoves = (row: number[], ship: number): [number,
number] => [
  row.indexOf(ship) - 1,
  row.lastIndexOf(ship) + 1

```

```

];

const getValidMoves = (
  expectedPlayer: string,
  boards: Boards,
  ship: number,
  [name, x, y]
): any[] => {
  const board = boards[expectedPlayer];
  const rowIndex = board.findIndex(r => r.some(c => c === ship));

  if (!isThereEnoughSpaceForNextMove(board, ship, x, y)) {
    return [];
  }
  if (rowIndex >= 0) {
    const row = board[rowIndex];
    const colIndex = row.findIndex(e => e === ship);

    const isHorizontal =
      row[colIndex - 1] === ship || row[colIndex + 1] === ship;
    if (isHorizontal) {
      const [left, right] = getTwoValidMoves(row, ship);
      return [{ x: rowIndex, y: left }, { x: rowIndex, y: right }
    ];
  }

  const isVertical =
    (board[rowIndex - 1] ? board[rowIndex - 1][colIndex] === s
hip : false) ||
    (board[rowIndex + 1] ? board[rowIndex + 1][colIndex] === s
hip : false);
  if (isVertical) {
    const [up, down] = getTwoValidMoves(
      board.map(r => r.filter((c, j) => j === colIndex)[0]),
      ship
    );
    return [{ x: up, y: colIndex }, { x: down, y: colIndex }];
  }

  return [
    { x: rowIndex, y: colIndex - 1 },
    { x: rowIndex, y: colIndex + 1 },
  ]
}

```

```

        { x: rowIndex - 1, y: colIndex },
        { x: rowIndex + 1, y: colIndex }
    ];
}

return [{ x: x, y: y }];
};

const isEmptyCell = (boards: Boards, [name, x, y]): boolean =>
    boards[name][x][y] === EMPTY;

const areSpacesAroundCellEmpty = (boards: Boards, [name, x, y]):
boolean =>
    (board =>
        (board[x - 1] && board[x - 1][y] === EMPTY) ||
        (board[x + 1] && board[x + 1][y] === EMPTY) ||
        board[x][y - 1] === EMPTY ||
        board[x][y + 1] === EMPTY)(boards[name]);

const canMove = (
    expectedPlayer: string,
    boards: Boards,
    ship: number,
    [name, x, y]
): boolean => {
    if (!isEmptyCell(boards, [name, x, y]) || name !== expectedPla
yer) {
        return false;
    }

    const validMoves = getValidMoves(expectedPlayer, boards, ship,
[name, x, y]);
    const isValidMove = validMoves.some(e => e.x === x && e.y === y
);

    return isValidMove;
};

const addShips$ = (player: string, boards: Boards) =>
    pipe(
        map((e: string) => e.split(',')),
        filter(e => e.length === 3),

```

```

    map(e => [e[0], parseInt(e[1]), parseInt(e[2])]),
    scan(
      (a, coords: any) => (
        (a.validMove =
          a.shipPartsLeft > 0
            ? canMove(player, boards, a.ship, coords)
            : isEmptyCell(boards, coords) &&
              (a.ship - 1 === 1 || areSpacesAroundCellEmpty(boards, coords))),
        a.validMove
          ? a.shipPartsLeft > 0
            ? (a.shipPartsLeft -= 1)
              : ((a.ship = a.ship - 1), (a.shipPartsLeft = a.ship - 1))
            : noop,
          (a.coords = coords),
          a
        ),
      { ship: 5, shipPartsLeft: 5, coords: [], validMove: true }
    ),
    filter(({ validMove }) => validMove),
    map(
      ({ ship, coords }) => (
        (boards[player][coords[1]][coords[2]] = ship), boards
      )
    ),
    paintBoards$,
    take(NUMBER_OF_SHIP_PARTS)
  );

const playerSetup$ = (boards: Boards) =>
  fromEvent(document, 'click').pipe(
    validClicks$,
    addShips$(PLAYER, boards)
  );

const computerSetup$ = (boards: Boards) =>
  interval().pipe(
    tap(i => (i % 70 === 0 ? (playerScoreContainer.innerHTML += ' ') : noop)),
    map(_ => `${COMPUTER}, ${random()}, ${random()}`),
    addShips$(COMPUTER, boards)
  );

```

```

    );

    const info$ = (container: HTMLElement, text: string) =>
      of({}).pipe(tap(_ => (container.innerHTML = text)));

    const createBoard = () =>
      Array(GAME_SIZE)
        .fill(EMPTY)
        .map(_ => Array(GAME_SIZE).fill(EMPTY));

    export const emptyBoards$ = of({
      [PLAYER]: createBoard(),
      [COMPUTER]: createBoard()
    });

    export const setup$ = (boards: Boards) =>
      concat(
        info$(computerScoreContainer, 'Setup your board!!!'),
        playerSetup$(boards),
        info$(playerScoreContainer, 'Computer setting up!!!'),
        computerSetup$(boards)
      );

```

game.ts

```

import { fromEvent, pipe, noop, Subject, BehaviorSubject, merge }
  from 'rxjs';
import { repeatWhen, delay, filter, map, takeWhile, tap } from '
  rxjs/operators';
import {
  GAME_SIZE,
  EMPTY,
  MISS,
  HIT,
  SHORTEST_SHIP,
  LONGEST_SHIP,
  PLAYER,
  COMPUTER,
  NUMBER_OF_SHIP_PARTS
} from './constants';

```

```
import { paintBoards, paintScores } from './html-renderer';
import { Boards, ComputerMove } from './interfaces';

export const random = () => Math.floor(Math.random() * Math.floor(
  GAME_SIZE));

export const validClicks$ = pipe(
  map((e: MouseEvent) => e.target['id']),
  filter(e => e)
);

const playerMove = new Subject();
const computerMove = new BehaviorSubject({ playerBoard: [], hits:
  {} });

const shot = (
  boards: Boards,
  player: string,
  x: number,
  y: number
): [number, number, boolean, number] =>
  ((boardValue): [number, number, boolean, number] => (
    (boards[player][x][y] = boardValue === EMPTY ? MISS : HIT),
    [x, y, boards[player][x][y] === HIT, boardValue]
  ))(boards[player][x][y]);

const isValidMove = (boards: Boards, player, x, y): boolean =>
  boards[player][x][y] !== HIT && boards[player][x][y] !== MISS;

const performShot$ = (
  boards: Boards,
  player: string,
  nextMove: (x, y, wasHit, boardValue) => void
) =>
  pipe(
    tap([player, x, y] =>
      !isValidMove(boards, player, x, y)
        ? nextMove(x, y, true, boards[player][x][y])
        : noop
    ),
    filter([player, x, y] => isValidMove(boards, player, x, y))
  ),
  ,
```

```
map(([_ , x, y]) => shot(boards, player, x, y)),
tap(
  ([x, y, wasHit, boardValue]) => (
    paintBoards(boards),
    nextMove(x, y, wasHit, boardValue),
    paintScores(computerScore$, playerScore$)
  )
);

const computerHits = (
  playerBoard: number[][],
  x: number,
  y: number,
  wasHit: boolean,
  boardValue: number
): ComputerMove => {
  if ([EMPTY, HIT, MISS].some(e => e === boardValue)) {
    return computerMove.value;
  }
  if (!computerMove.value.hits[boardValue]) {
    computerMove.value.hits[boardValue] = [];
  }
  computerMove.value.hits[boardValue].push({ x, y });
  computerMove.value.playerBoard = playerBoard;

  return computerMove.value;
};

const nextComputerMove = (): [string, number, number] => {
  const hits = computerMove.value.hits;
  const shipToPursue = Object.keys(hits).find(
    e => hits[e].length !== parseInt(e)
  );
  if (!shipToPursue) {
    return [PLAYER, random(), random()];
  }

  const playerBoard = computerMove.value.playerBoard;
  const shipHits = hits[shipToPursue];
  if (shipHits.length === 1) {
    const hit = shipHits[0];
```

```

const shotCandidates = [
  [hit.x, hit.y - 1],
  [hit.x, hit.y + 1],
  [hit.x - 1, hit.y],
  [hit.x + 1, hit.y]
].filter(
  ([x, y]) =>
    playerBoard[x] &&
    playerBoard[x][y] !== undefined &&
    playerBoard[x][y] !== MISS &&
    playerBoard[x][y] !== HIT
);

return [PLAYER, shotCandidates[0][0], shotCandidates[0][1]];
}

const getOrderedHits = key =>
  (orderedHits => [orderedHits[0], orderedHits[orderedHits.length - 1]])(
    shipHits.sort((h1, h2) => (h1[key] > h2[key] ? 1 : -1))
  );
const isHorizontal = shipHits.every(e => e.x === shipHits[0].x)
;

if (isHorizontal) {
  const [min, max] = getOrderedHits('y');
  return [
    PLAYER,
    min.x,
    playerBoard[min.x][min.y - 1] !== undefined &&
    playerBoard[min.x][min.y - 1] !== HIT &&
    playerBoard[min.x][min.y - 1] !== MISS
      ? min.y - 1
      : max.y + 1
  ];
}

const [min, max] = getOrderedHits('x');
return [
  PLAYER,
  playerBoard[min.x - 1] !== undefined &&

```



```

    playerBoard[min.x - 1][min.y] !== HIT &&
    playerBoard[min.x - 1][min.y] !== MISS
      ? min.x - 1
      : max.x + 1,
    min.y
  ];
};

const initialScore = () => ({
  score: 0,
  ships: { 5: 5, 4: 4, 3: 3, 2: 2, 1: 1 }
});
export const playerScore$ = new BehaviorSubject(initialScore());
export const computerScore$ = new BehaviorSubject(initialScore());
;
export const isNotGameOver = _ =>
  computerScore$.value.score < NUMBER_OF_SHIP_PARTS &&
  playerScore$.value.score < NUMBER_OF_SHIP_PARTS;

const scoreChange = (subject: BehaviorSubject<any>, boardValue:
number) =>
  boardValue >= SHORTEST_SHIP && boardValue <= LONGEST_SHIP
    ? ((subject.value.ships[boardValue] -= 1),
      subject.next({
        score: subject.value.score + 1,
        ships: subject.value.ships
      }))
    : noop;

const computerShot$ = (boards: Boards) =>
  computerMove.pipe(
    delay(200),
    map(_ => nextComputerMove()),
    performShot$(boards, PLAYER, (x, y, wasHit, boardValue) =>
      wasHit
        ? (scoreChange(computerScore$, boardValue),
          computerMove.next(
            computerHits(boards[PLAYER], x, y, wasHit, boardValue
          )
        )
      )
    : playerMove.next()
  )

```

```

    );

    const playerShot$ = (boards: Boards) =>
      fromEvent(document, 'click').pipe(
        validClicks$,
        map((click: string) => click.split(',')),
        filter(([player]) => player === COMPUTER),
        performShot$(boards, COMPUTER, (x, y, wasHit, boardValue) =>
          wasHit
            ? scoreChange(playerScore$, boardValue)
            : computerMove.next(computerMove.value)
        ),
        takeWhile(([x, y, wasHit]) => wasHit),
        repeatWhen(_ => playerMove)
      );

    export const shots$ = (boards: Boards) =>
      merge(playerShot$(boards), computerShot$(boards));

```

html-renderer.ts

```

import { BehaviorSubject, pipe } from "rxjs";
import { tap } from "rxjs/operators";
import {
  NUMBER_OF_SHIP_PARTS,
  EMPTY,
  MISS,
  HIT,
  PLAYER,
  COMPUTER
} from "./constants";
import { Boards } from "./interfaces";

const byId = (id: string): HTMLElement => document.getElementById(
  id
);
export const computerScoreContainer = byId("computer_score");
export const playerScoreContainer = byId("player_score");

const playerCells = (cell: number): string | number =>
  cell !== EMPTY ? (cell === MISS ? "o" : cell === HIT ? "x" : c

```

```

    ell) : "_";
    const computerCells = (cell: number): string | number =>
        cell === HIT || cell === MISS ? (cell === MISS ? "o" : "x") :
        "_";

    export const paintBoard = (
        container: HTMLElement,
        playerName: string,
        board: number[][]
    ) => (
        (container.innerHTML = ""),
        board.forEach((r, i) =>
            r.forEach(
                (c, j) =>
                    (container.innerHTML += `
                        <div id=${playerName},${i},${j}
                        style='float:left; margin-left: 5px'>
                        ${playerName === PLAYER ? playerCells(c) : computerCells(c)}
                        </div>`),
                    (container.innerHTML += "<br/>")
                )
            ),
        (container.innerHTML += "<br/><br/>")
    );

    export const paintShipsInfo = (scoreSubject: BehaviorSubject<any>
    ) =>
        Object.keys(scoreSubject.value.ships).reduce(
            (a, c) => ((a += `<b>${c} </b>: ${scoreSubject.value.ships[c]
            } | `), a),
            ""
        );

    export const paintScores = (
        computerScore: BehaviorSubject<any>,
        playerScore: BehaviorSubject<any>
    ) =>
        ((c: HTMLElement, p: HTMLElement) => (
            (c.innerHTML = ""),
            (c.innerHTML += "Computer score: " + computerScore.value.sco
            re + "<br/>"),
            paintShipsInfo(computerScore),

```

```

    (c.innerHTML += "Ships: " + paintShipsInfo(computerScore)),
    (p.innerHTML = ""),
    (p.innerHTML += "Player score: " + playerScore.value.score +
    "<br/>"),
    (p.innerHTML += "Ships: " + paintShipsInfo(playerScore))
  ))(computerScoreContainer, playerScoreContainer);

export const paintBoards = (boards: Boards) => (
  paintBoard(byId("player_board"), PLAYER, boards[PLAYER]),
  paintBoard(byId("computer_board"), COMPUTER, boards[COMPUTER])
);

export const paintBoards$ = pipe<any, any>(tap(paintBoards));

export const displayGameOver = (computerScore: BehaviorSubject<any>) => () => {
  const gameOverText = `GAME OVER,
    ${
      computerScore.value.score === NUMBER_OF_SHIP_PARTS
        ? "Computer"
        : "Player"
    }
    won`;
  playerScoreContainer.innerHTML = gameOverText;
  computerScoreContainer.innerHTML = gameOverText;
};

```

interfaces.ts

```

export interface Boards {
  player: [string, number[][]];
  computer: [string, number[][]];
}

export interface ComputerMove {
  playerBoard: number[];
  hits: {};
}

```

constants.ts

```
export const GAME_SIZE = 12;
export const NUMBER_OF_SHIP_PARTS = 15;
export const EMPTY = 0;
export const MISS = 8;
export const HIT = 9;
export const SHORTEST_SHIP = 1;
export const LONGEST_SHIP = 5;
export const PLAYER = 'p';
export const COMPUTER = 'c';
```

Operators Used

- [concat](#)
- [delay](#)
- [filter](#)
- [finalize](#)
- [fromEvent](#)
- [interval](#)
- [map](#)
- [merge](#)
- [of](#)
- [repeatWhen](#)
- [scan](#)
- [switchMap](#)
- [take](#)
- [takeWhile](#)
- [tap](#)


Subjects Used

- [BehaviorSubject](#)
- [Subject](#)

Breakout Game

By [adamlubek](#)


This recipe demonstrates an RxJS implementation of Breakout game.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))

Score: 0 Lives: 3



index.ts

```
// RxJS v6+
import { fromEvent, of, interval, combineLatest, generate, noop }
  from 'rxjs';
import { map, mergeMap, pluck, startWith, scan, toArray, takeWhile, tap } from 'rxjs/operators';
import { gameSize } from './constants';
import { Player, Ball, GameObject } from './interfaces';
import { render } from './html-renderer';

const createGameObject = (x, y) => ({ x, y });

const player$ = combineLatest(
  of({ ...createGameObject(gameSize - 2, (gameSize / 2) - 1), score: 0, lives: 3 }),
  fromEvent(document, 'keyup').pipe(startWith({ code: '' }), pluck('code'))
).pipe(
  map(([player, key]) => (
    key === 'ArrowLeft'
      ? player.y -= 1
      : key === 'ArrowRight'
        ? player.y += 1
        : noop
    , player)
  )
);

const ball$ = combineLatest(
  of({ ...createGameObject(gameSize / 2, (gameSize - 3)), dirX: 1, dirY: 1 }),
  interval(150)
).pipe(
  map(([ball, _]: [Ball, number]) => (
    ball.dirX *= ball.x > 0 ? 1 : -1,
    ball.dirY *= (ball.y > 0 && ball.y < gameSize - 1) ? 1 : -1,
    ball.x += 1 * ball.dirX,
    ball.y -= 1 * ball.dirY,
    ball)
  )
);
```



```

const bricks$ = generate(1, x => x < 8, x => x + 1)
  .pipe(
    mergeMap(r => generate(r % 2 === 0 ? 1 : 0, x => x < gameSize
, x => x + 2)
    .pipe(map(c => createGameObject(r, c)))
  ),
  toArray()
)

const processGameCollisions = (_, [player, ball, bricks]: [Player
, Ball, GameObject[]])
: [Player, Ball, GameObject[]] => (
  (collidingBrickIndex => collidingBrickIndex > -1
    ? (bricks.splice(collidingBrickIndex, 1), ball.dirX *= -1,
player.score++)
    : noop
  )(bricks.findIndex(e => e.x === ball.x && e.y === ball.y)),
  ball.dirX *= player.x === ball.x && player.y === ball.y ? -1
: 1,
  ball.x > player.x ? (player.lives-- , ball.x = (gameSize / 2)
- 3) : noop,
  [player, ball, bricks]
)

combineLatest(player$, ball$, bricks$)
  .pipe(
    scan<[Player, Ball, GameObject[]], [Player, Ball, GameObject[
]]>(processGameCollisions),
    tap(render),
    takeWhile(([player]: [Player, Ball, GameObject[]]) => player.
lives > 0)
  ).subscribe()

```

interfaces.ts

```
export interface GameObject {  
  x: number;  
  y: number;  
}  
export interface Player extends GameObject {  
  score: number;  
  lives: number;  
}  
export interface Ball extends GameObject {  
  dirX: number;  
  dirY: number;  
}
```

constants.ts

```
export const gameSize = 20;
```

html-renderer.ts

```
import { gameSize } from './constants';  
import { Player, Ball, GameObject } from './interfaces';  
  
const empty = 0;  
const player = 1;  
const ball = 2;  
const brick = 3;  
  
const createElem = col => {  
  const elem = document.createElement('div');  
  elem.classList.add('board');  
  elem.style.display = 'inline-block';  
  elem.style.marginLeft = '10px';  
  elem.style.height = '6px';  
  elem.style.width = '6px';  
  elem.style['background-color'] =  
    col === empty  
      ? 'white'  
      : col === player  
        ? 'cornflowerblue'
```

```
      : col === b11
      ? 'gray'
      : 'silver';
    elem.style['border-radius'] = col === b11 ? '100%' : '0%';
    return elem;
  };

export const render = ([player, ball, bricks]: [
  Player,
  Ball,
  GameObject[]
]) => {
  const game = Array(gameSize)
    .fill(0)
    .map(e => Array(gameSize).fill(0));
  game[player.x][player.y] = player;
  game[ball.x][ball.y] = ball;
  bricks.forEach(b => (game[b.x][b.y] = brick));

  document.body.innerHTML = `Score: ${player.score} Lives: ${pla
yer.lives} <br/>`;
  game.forEach(r => {
    const rowContainer = document.createElement('div');
    r.forEach(c => rowContainer.appendChild(createElem(c)));
    document.body.appendChild(rowContainer);
  });
};
```

Operators Used


- `combineLatest`
- `fromEvent`
- `generate`
- `interval`
- `mergeMap`
- `of`
- `pluck`
- `scan`
- `startWith`

- `takeWhile`
- `tap`
- `toArray`

Car Racing Game

By [adamlubek](#)


This recipe demonstrates RxJS implementation of Car Racing game.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))

index.ts

```
// RxJS v6+
import {
  interval,
  fromEvent,
  combineLatest,
  of,
  BehaviorSubject,
  noop
} from 'rxjs';
import {
  scan,
  tap,
  pluck,
  startWith,
  takeWhile,
  finalize,
  switchMap
} from 'rxjs/operators';
import { Car, Road, Player, Game } from './interfaces';
import { gameHeight, gameWidth, levelDuration } from './constants';
import { updateState } from './state';
import { render, renderGameOver } from './html-renderer';

const car = (x: number, y: number): Car => ({ x, y, scored: false });
const randomCar = (): Car =>
  car(0, Math.floor(Math.random() * Math.floor(gameWidth)));
const gameSpeed$ = new BehaviorSubject(200);

const road$ = gameSpeed$.pipe(
  switchMap(i =>
    interval(i).pipe(
      scan(
        (road: Road, _: number): Road => (
          (road.cars = road.cars.filter(c => c.x < gameHeight - 1)
        ),
        road.cars[0].x === gameHeight / 2
          ? road.cars.push(randomCar())
          : noop,
        road.cars.forEach(c => c.x++),
      )
    )
  )
);
```

```
        road
      ),
      { cars: [randomCar()] }
    )
  )
);

const keys$ = fromEvent(document, 'keyup').pipe(
  startWith({ code: '' }),
  pluck('code')
);

const player$ = keys$.pipe(
  scan(
    (player: Player, key: string): Player => (
      (player.y +=
        key === 'ArrowLeft' && player.y > 0
          ? -1
          : key === 'ArrowRight' && player.y < gameWidth - 1
            ? 1
            : 0),
      player
    ),
    { y: 0 }
  )
);

const state$ = of({
  score: 1,
  lives: 3,
  level: 1,
  duration: levelDuration,
  interval: 200
});

const isNotGameOver = ([state]: Game) => state.lives > 0;

const game$ = combineLatest(state$, road$, player$).pipe(
  scan(updateState(gameSpeed$)),
  tap(render),
  takeWhile(isNotGameOver),
```

```
    finalize(renderGameOver)  
  );  
  
  game$.subscribe();
```

state.ts


```
import { BehaviorSubject, noop } from 'rxjs';
import { Game } from './interfaces';
import { gameHeight, gameWidth, levelDuration } from './constants';

const handleScoreIncrease = ([state, road, player]: Game) =>
  !road.cars[0].scored &&
  road.cars[0].y !== player.y &&
  road.cars[0].x === gameHeight - 1
  ? ((road.cars[0].scored = true), (state.score += 1))
  : noop;

const handleCollision = ([state, road, player]: Game) =>
  road.cars[0].x === gameHeight - 1 && road.cars[0].y === player.y
  ? (state.lives -= 1)
  : noop;

const updateSpeed = ([state]: Game, gameSpeed: BehaviorSubject<number>) => (
  (state.duration -= 10),
  state.duration < 0
  ? ((state.duration = levelDuration * state.level),
    state.level++,
    (state.interval -= state.interval > 60 ? 20 : 0),
    gameSpeed.next(state.interval))
  : () => {}
);

export const updateState = (gameSpeed: BehaviorSubject<number>)
=> (
  —
  game: Game
) => (
  handleScoreIncrease(game),
  handleCollision(game),
  updateSpeed(game, gameSpeed),
  game
);
```

html-renderer.ts

```

import { Game } from './interfaces';
import { gameHeight, gameWidth, car, player } from './constants';

const createElem = (column: number) =>
  (elem => (
    (elem.style.display = 'inline-block'),
    (elem.style.marginLeft = '3px'),
    (elem.style.height = '12px'),
    (elem.style.width = '6px'),
    (elem.style.borderRadius = '40%'),
    (elem.style['background-color'] =
      column === car ? 'green' : column === player ? 'blue' : 'white'),
    elem
  ))(document.createElement('div'));

export const render = ([state, road, playerPosition]: Game) =>
  (renderFrame => (
    road.cars.forEach(c => (renderFrame[c.x][c.y] = car)),
    (document.getElementById(
      'game'
    ).innerHTML = `Score: ${state.score} Lives: ${state.lives} Level: ${state.level}`),
    (renderFrame[gameHeight - 1][playerPosition.y] = player),
    renderFrame.forEach(r => {
      const rowContainer = document.createElement('div');
      r.forEach(c => rowContainer.appendChild(createElem(c)));
      document.getElementById('game').appendChild(rowContainer);
    })
  ))(
    Array(gameHeight)
      .fill(0)
      .map(e => Array(gameWidth).fill(0))
  );

export const renderGameOver = () =>
  (document.getElementById('game').innerHTML += '<br/>GAME OVER!!!');

```

interfaces.ts

```
export interface Car {
  x: number;
  y: number;
  scored: boolean;
}

export interface Road {
  cars: Car[];
}

export interface State {
  score: number;
  lives: number;
  level: number;
  duration: number;
  interval: number;
}

export interface Player {
  y: number;
}

export type Game = [State, Road, Player];
```

constants.ts

```
export const gameHeight = 10;
export const gameWidth = 6;

export const levelDuration = 500;

export const car = 1;
export const player = 2;
```

index.html

```
<style>
  .road {
    width: 100px;
    height: 180px;
    margin-top: 25px;
    overflow: hidden;
    position: absolute;
  }

  .dotted {
    margin-top: -100px;
    height: 300px;
    border-left: 2px dashed lightgray;
    position: absolute;
    animation: road-moving 1s infinite linear;
  }

  @keyframes road-moving {
    100% {
      transform: translateY(100px);
    }
  }
</style>

<div class="road">
  <div class="dotted" style="margin-left: 0px;"></div>
  <div class="dotted" style="margin-left: 9px;"></div>
  <div class="dotted" style="margin-left: 18px;"></div>
  <div class="dotted" style="margin-left: 27px;"></div>
  <div class="dotted" style="margin-left: 36px;"></div>
  <div class="dotted" style="margin-left: 45px;"></div>
  <div class="dotted" style="margin-left: 54px;"></div>
</div>
<div id="game"></div>
```

Operators Used


- [BehaviorSubject](#)
- [combineLatest](#)

- [finalize](#)
- [fromEvent](#)
- [interval](#)
- [of](#)
- [pluck](#)
- [scan](#)
- [startWith](#)
- [switchMap](#)
- [takeWhile](#)
- [tap](#)

Catch The Dot Game

By [adamlubek](#)


This recipe shows usage of scan operator for state management in simple game

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))



index.ts

```
// RxJS v6+
import { fromEvent, interval } from 'rxjs';
import { tap, scan, map, switchMap, takeWhile } from 'rxjs/operators';
import {
  dot,
  updatedDot,
  setTimerText,
  resetDotSize,
  moveDot
} from './dom-updater';

interface State {
  score: number;
  intrvl: number;
}

const makeInterval = (val: State) =>
  interval(val.intrvl).pipe(
    map(v => 5 - v),
    tap(setTimerText)
  );

const gameState: State = { score: 0, intrvl: 500 };
const nextState = (acc: State) => ({
  score: (acc.score += 1),
  intrvl: acc.score % 3 === 0 ? (acc.intrvl -= 50) : acc.intrvl
});

const isNotGameOver = intervalValue => intervalValue >= 0;

const game$ = fromEvent(dot, 'mouseover').pipe(
  tap(moveDot),
  scan < Event,
  State > (nextState, gameState),
  tap(state => updatedDot(state.score)),
  switchMap(makeInterval),
  tap(resetDotSize),
  takeWhile(isNotGameOver)
);

game$.subscribe(n => {}, e => {}, () => setTimerText('ouch!'));
```


dom-updater.ts

```
const random = () => Math.random() * 300;
const elem = id => document.getElementById(id);
const setElementText = (elem, text) => (elem.innerText = text.toString());
const timer = elem('timer');
const setDotSize = size => {
  dot.style.height = `${size}px`;
  dot.style.width = `${size}px`;
};

export const dot = elem('dot');
export const updatedDot = score => {
  if (score % 3 === 0) {
    dot.style.backgroundColor =
      '#' + ((Math.random() * 0xffffffff) << 0).toString(16);
  }
  setElementText(dot, score);
};
export const setTimerText = text => setElementText(timer, text);
export const moveDot = () => {
  setDotSize(5);
  dot.style.transform = `translate(${random()}px, ${random()}px)`;
};
export const resetDotSize = () => setDotSize(30);
```

html

```
<style>
  #dot {
    margin-top: 10px;
    height: 30px;
    width: 30px;
    background-color: lightgray;
    border-radius: 50%;
    transition: all 0.6s ease-in-out;
    text-align: center;
    color: white;
  }

  #timer {
    position: absolute;
    top: 150px;
    left: 150px;
    opacity: 0.1;
    font-size: 60px;
  }
</style>


<div id="timer"></div>
<div id="dot"></div>
```

- [fromEvent](#)
- [interval](#)
- [map](#)
- [scan](#)
- [switchMap](#)
- [takeWhile](#)
- [tap](#)

Click Ninja Game

By [adamlubek](#)


This recipe shows usage of time interval operator in a simple game

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))

index.ts

```
// RxJS v6+
import { fromEvent, TimeInterval } from 'rxjs';
import { setInterval, takeWhile, scan, tap, repeat, finalize }
from 'rxjs/operators';
import { render, clear } from './html-renderer';

interface State {
  score: number;
  interval: number;
  threshold: number;
}

fromEvent(document, 'mousedown').pipe(
  setInterval(),
  scan<TimeInterval<Event>, State>((state, timeInterval) => ({
    score: state.score + 1,
    interval: timeInterval.interval,
    threshold: state.threshold - 2
  })), { score: 0, interval: 0, threshold: 300 }),
  takeWhile((state: State) => state.interval < state.threshold),
  tap((state: State) => render(state.score, Math.floor(state.score / 10))),
  finalize(clear),
  repeat()
).subscribe();
```

html-renderer.ts

```
const texts = {
  0: 'click, click',
  1: 'keep clicking',
  2: 'wow',
  3: 'not tired yet?!',
  4: 'click master!',
  5: 'inhuman!!!',
  6: 'ininhuman!!!'
};

const text = (score: number, level: number) => `${texts[level]}
  \n ${score}`;
```

```
export const render = (score: number, level: number) => {
  const id = 'level' + level;
  const element = document.getElementById(id);
  const innerText = text(score, level);
  if (element) {
    element.innerText = innerText;
  } else {
    const elem = document.createElement('div');
    elem.id = id;
    elem.style.zIndex = `${level}`;
    elem.style.position = 'absolute';
    elem.style.height = '150px';
    elem.style.width = '150px';
    elem.style.borderRadius = '10px';
    const position = level * 20;
    elem.style.top = position + 'px';
    elem.style.left = position + 'px';
    const col = 100 + position;
    elem.style.background = `rgb(0,${col},0)`;
    elem.style.color = 'white';
    elem.innerText = innerText;
    elem.style.textAlign = 'center';
    elem.style.verticalAlign = 'middle';
    elem.style.lineHeight = '90px';
    document.body.appendChild(elem);
  }
};

export const clear = () => (document.body.innerText = '');
```

html

```
<div>How fast can you click?!</div>
```


- [finalize](#)
- [fromEvent](#)
- [repeat](#)
- [scan](#)

- `takeWhile`
- `tap`
- `timeInterval`

Flappy Bird Game

By [adamlubek](#)


This recipe demonstrates RxJs implementation of Flappy Bird like game.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →

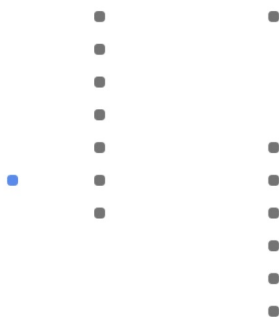


with Brian Troncone
Creator of **LearnRxJS.io**

Example Code

([StackBlitz](#))

Lives: 3, Score: 3



index.ts

```
// RxJS v6+
import { interval, merge, combineLatest, fromEvent } from 'rxjs';

import { tap, scan, takeWhile } from 'rxjs/operators';
import { paint } from './html-renderer';

const gamePipe = (x, y) => ({ x, y, checked: false });
const gameSize = 10;
const createPipes = y =>
  (random =>
    Array.from(Array(gameSize).keys())
      .map(e => gamePipe(e, y))
      .filter(e => e.x < random || e.x > random + 2))(
    Math.floor(Math.random() * Math.floor(gameSize))
  );

const gamePipes$ = interval(500).pipe(
  scan < any,
  any >
  (acc =>
    (acc.length < 2 ? [...acc, createPipes(gameSize)] : acc)
      .filter(c => c.some(e => e.y > 0))
      .map(cols => cols.map(e => gamePipe(e.x, e.y - 1))),
    [createPipes(gameSize / 2), createPipes(gameSize)])
);

const fly = xPos => (xPos > 0 ? (xPos -= 1) : xPos);
const fall = xPos => (xPos < gameSize - 1 ? (xPos += 1) : gameSi
ze - 1);
const bird$ = merge(interval(300), fromEvent(document, 'keydown'))
  .pipe(
    scan < any,
    any >
    ((xPos, curr) => (curr instanceof KeyboardEvent ? fly(xPos) :
      fall(xPos)),
      gameSize - 1)
  );

const updateGame = (bird, pipes) =>
  (game => (
    pipes.forEach(col => col.forEach(v => (game[v.x][v.y] = 2))),
```



```

    (game[bird][0] = 1),
    game
  ))(
    Array(gameSize)
      .fill(0)
      .map(e => Array(gameSize).fill(0))
  );

const valueOnCollisionFor = pipes => ({
  when: predicate =>
    !pipes[0][0].checked && predicate ? ((pipes[0][0].checked =
true), 1) : 0
});

combineLatest(bird$, gamePipes$)
  .pipe(
    scan < any,
    any >
    ((state, [bird, pipes]) => ({
      bird: bird,
      pipes: pipes,
      lives:
        state.lives -
        valueOnCollisionFor(pipes).when(
          pipes.some(c => c.some(c => c.y === 0 && c.x === bird
))
    ),
      score:
        state.score + valueOnCollisionFor(pipes).when(pipes[0][
0].y === 0)
    })),
    { lives: 3, score: 0, bird: 0, pipes: [] }),
    tap(state =>
      paint(updateGame(state.bird, state.pipes), state.lives, st
ate.score)
    ),
    takeWhile(state => state.lives > 0)
  )
  .subscribe();

```

html-renderer.ts

```
const createElem = col => {
  const elem = document.createElement('div');
  elem.classList.add('board');
  elem.style.display = 'inline-block';
  elem.style.marginLeft = '10px';
  elem.style.height = '6px';
  elem.style.width = '6px';
  elem.style['background-color'] =
    col === 0
      ? 'white'
      : col === 1
      ? 'cornflowerblue'
      : col === 2
      ? 'gray'
      : 'silver';
  elem.style['border-radius'] = '90%';
  return elem;
};

export const paint = (game: number[][], lives, score) => {
  document.body.innerHTML = `Lives: ${lives}, Score: ${score}`;

  game.forEach(row => {
    const rowContainer = document.createElement('div');
    row.forEach(col => rowContainer.appendChild(createElem(col)))
  });
  document.body.appendChild(rowContainer);
};
```

Operators Used


- [combineLatest](#)
- [fromEvent](#)
- [interval](#)
- [merge](#)
- [scan](#)

- [takeWhile](#)
- [tap](#)

Game Loop

By [@barryrowe](#)


This recipe demonstrates one way you might create a Game Loop as a combined set of streams. The recipe is intended to highlight how you might re-think existing problems with a reactive approach. In this recipe we provide the overall loop as a stream of frames and their deltaTimes since the previous frames. Combined with this is a stream of user inputs, and the current gameState, which we can use to update our objects, and render to to the screen on each frame emission.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

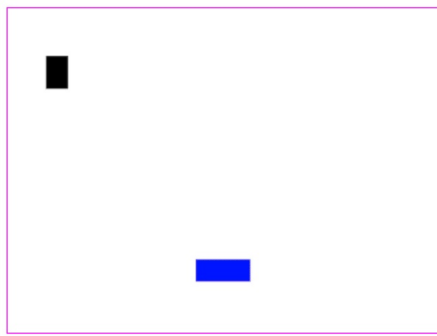
Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))



60

Each time a block hits a wall, it gets faster. You can hit SPACE to pause the boxes. They will change colors to show they are paused.

```
import { BehaviorSubject, Observable, of, fromEvent } from 'rxjs'
;
import { buffer, bufferCount, expand, filter, map, share, tap,
withLatestFrom } from 'rxjs/operators';

import { IFrameData } from './frame.interface';
import { KeyUtil } from './keys.util';
import { clampMag, runBoundaryCheck, clampTo30FPS } from './game
.util';

const boundaries = {
  left: 0,
  top: 0,
  bottom: 300,
  right: 400
};

const bounceRateChanges = {
  left: 1.1,
  top: 1.2,
  bottom: 1.3,
  right: 1.4
}

const baseObjectVelocity = {
  x: 30,
  y: 40,
```

```
    maxX: 250,
    maxY: 200
};

const gameArea: HTMLElement = document.getElementById('game');
const fps: HTMLElement = document.getElementById('fps');

/**
 * This is our core game loop logic. We update our objects and gameState here
 * each frame. The deltaTime passed in is in seconds, we are given our current state,
 * and any inputStates. Returns the updated Game State
 */
const update = (deltaTime: number, state: any, inputState: any): any => {
  //console.log("Input State: ", inputState);
  if(state['objects'] === undefined) {
    state['objects'] = [
      {
        // Transformation Props
        x: 10, y: 10, width: 20, height: 30,
        // State Props
        isPaused: false, toggleColor: '#FF0000', color: '#000000'
      },
      {
        // Movement Props
        velocity: baseObjectVelocity
      },
      {
        // Transformation Props
        x: 200, y: 249, width: 50, height: 20,
        // State Props
        isPaused: false, toggleColor: '#00FF00', color: '#0000FF'
      },
      {
        // Movement Props
        velocity: {x: -baseObjectVelocity.x, y: 2*baseObjectVelocity.y}
      }
    ];
  } else {

    state['objects'].forEach((obj) => {
      // Process Inputs
```

```
if (inputState['spacebar']) {
  obj.isPaused = !obj.isPaused;
  let newColor = obj.toggleColor;
  obj.toggleColor = obj.color;
  obj.color = newColor;
}

// Process GameLoop Updates
if(!obj.isPaused) {

  // Apply Velocity Movements
  obj.x = obj.x += obj.velocity.x*deltaTime;
  obj.y = obj.y += obj.velocity.y*deltaTime;

  // Check if we exceeded our boundaries
  const didHit = runBoundaryCheck(obj, boundaries);
  // Handle boundary adjustments
  if(didHit){
    if(didHit === 'right' || didHit === 'left') {
      obj.velocity.x *= -bounceRateChanges[didHit];
    } else {
      obj.velocity.y *= -bounceRateChanges[didHit];
    }
  }
}

// Clamp Velocities in case our boundary bounces have gotten
// us going toooooo fast.
obj.velocity.x = clampMag(obj.velocity.x, 0, baseObjectVelocity.maxX);
obj.velocity.y = clampMag(obj.velocity.y, 0, baseObjectVelocity.maxY);
});
}

return state;
}

/**
 * This is our rendering function. We take the given game state
 * and render the items
```

```

    * based on their latest properties.
    */
const render = (state: any) => {
    const ctx: CanvasRenderingContext2D = (<HTMLCanvasElement>game
Area).getContext('2d');
    // Clear the canvas
    ctx.clearRect(0, 0, gameArea.clientWidth, gameArea.clientHeight);

    // Render all of our objects (simple rectangles for simplicity)

    state['objects'].forEach((obj) => {
        ctx.fillStyle = obj.color;
        ctx.fillRect(obj.x, obj.y, obj.width, obj.height);
    });
};

/**
 * This function returns an observable that will emit the next f
rame once the
 * browser has returned an animation frame step. Given the previ
ous frame it calculates
 * the delta time, and we also clamp it to 30FPS in case we get
long frames.
 */
const calculateStep: (prevFrame: IFrameData) => Observable<IFram
eData> = (prevFrame: IFrameData) => {
    return Observable.create((observer) => {

        requestAnimationFrame((frameStartTime) => {
            // Millis to seconds
            const deltaTime = prevFrame ? (frameStartTime - prevFrame.
frameStartTime)/1000 : 0;
            observer.next({
                frameStartTime,
                deltaTime
            });
        })
    })
    .pipe(

```



```
    map(clampTo30FPS)
  )
};

// This is our core stream of frames. We use expand to recursive
// ly call the
// `calculateStep` function above that will give us each new Fr
// ame based on the
// window.requestAnimationFrame calls. Expand emits the value o
// f the called functions
// returned observable, as well as recursively calling the func
// tion with that same
// emitted value. This works perfectly for calculating our fram
// e steps because each step
// needs to know the lastStepFrameTime to calculate the next. W
// e also only want to request
// a new frame once the currently requested frame has returned.
const frames$ = of(undefined)
  .pipe(
    expand((val) => calculateStep(val)),
    // Expand emits the first value provided to it, and in this
    // case we just want to ignore the undefined input frame
    filter(frame => frame !== undefined),
    map((frame: IFrameData) => frame.deltaTime),
    share()
  )

// This is our core stream of keyDown input events. It emits an
// object like `{"spacebar": 32}`
// each time a key is pressed down.
const keyDown$ = fromEvent(document, 'keydown')
  .pipe(
    map((event: KeyboardEvent) => {
      const name = KeyUtil.codeToKey(''+event.keyCode);
      if (name !== ''){
        let keyMap = {};
        keyMap[name] = event.code;
        return keyMap;
      } else {
        return undefined;
      }
    })
  ),
```

```
    filter((keyMap) => keyMap !== undefined)
  );

// Here we buffer our keyDown stream until we get a new frame emission. This
// gives us a set of all the keyDown events that have triggered since the previous
// frame. We reduce these all down to a single dictionary of keys that were pressed.
const keysDownPerFrame$ = keysDown$
  .pipe(
    buffer(frames$),
    map((frames: Array<any>) => {
      return frames.reduce((acc, curr) => {
        return Object.assign(acc, curr);
      }, {});
    })
  );

// Since we will be updating our gamestate each frame we can use an Observable
// to track that as a series of states with the latest emission being the current
// state of our game.
const gameState$ = new BehaviorSubject({});

// This is where we run our game!
// We subscribe to our frames$ stream to kick it off, and make sure to
// combine in the latest emission from our inputs stream to get the data
// we need to perform our gameState updates.
frames$
  .pipe(
    withLatestFrom(keysDownPerFrame$, gameState$),
    // HOMEWORK_OPPORTUNITY: Handle Key-up, and map to a true KeyState change object
    map(([deltaTime, keysDown, gameState]) => update(deltaTime, gameState, keysDown)),
    tap((gameState) => gameState$.next(gameState))
  )
)
```

```
.subscribe((gameState) => {
  render(gameState);
});

// Average every 10 Frames to calculate our FPS
frames$
  .pipe(
    bufferCount(10),
    map((frames) => {
      const total = frames
        .reduce((acc, curr) => {
          acc += curr;
          return acc;
        }, 0);

      return 1/(total/frames.length);
    })
  ).subscribe((avg) => {
    fps.innerHTML = Math.round(avg) + ' ';
  })
```

supporting js

- [game.util.ts](#)
- [keys.util.ts](#)
- [frame.interface.ts](#)

html

```
<canvas width="400px" height="300px" id="game"></canvas>
<div id="fps"></div>
<p class="instructions">
  Each time a block hits a wall, it gets faster. You can hit SPA
  CE to pause the
  boxes. They will change colors to show they are paused.
</p>
```

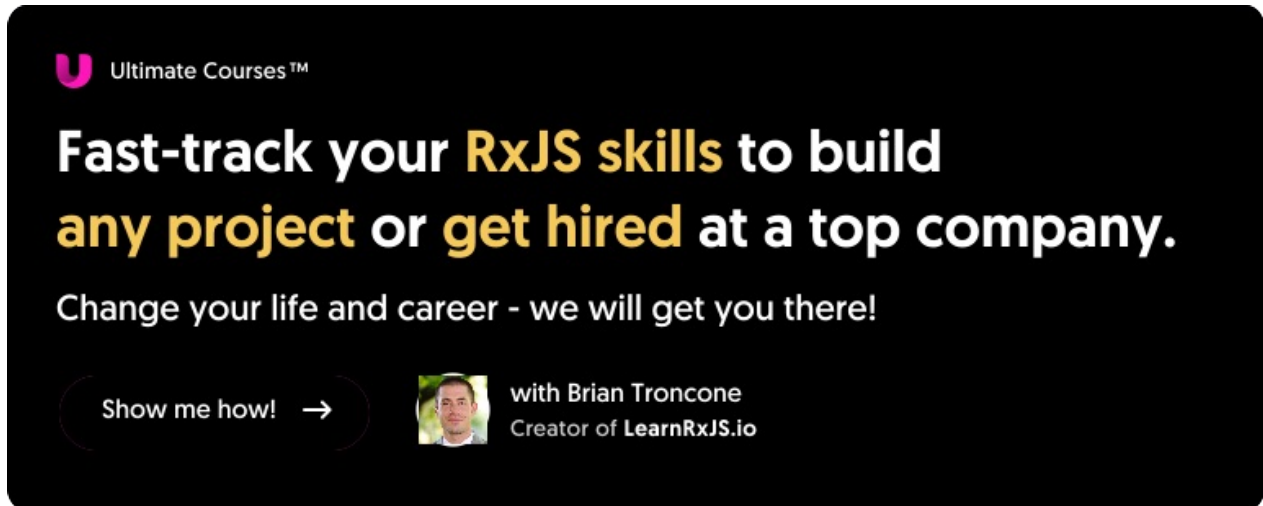
Operators Used

- [buffer](#)
- [bufferCount](#)
- [expand](#)
- [filter](#)
- [fromEvent](#)
- [map](#)
- [share](#)
- [tap](#)
- [withLatestFrom](#)

Horizontal scroll indicator

By [adamlubek](#)

This recipe demonstrates the creation of a horizontal scroll indicator.



Example Code

([StackBlitz](#))

```
// RxJS v6+
import { fromEvent } from 'rxjs';
import { throttleTime, tap } from 'rxjs/operators';

const scrollIndication = document.getElementById('indication');
const getScrollWidth = () => {
  const doc = document.documentElement;
  // https://www.w3schools.com/howto/howto_js_scroll_indicator.asp
  const winScroll = doc.scrollTop;
  const height = doc.scrollHeight - doc.clientHeight;

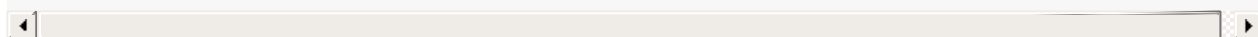
  return (winScroll / height) * 100;
};
const setScroll = _ => (scrollIndication.style.width = getScrollWidth() + '%');

fromEvent(document, 'scroll')
  .pipe(
    throttleTime(20),
    tap(setScroll)
  )
  .subscribe();
```

html

```
<style>
  #indication {
    position: fixed;
    width: 5px;
    height: 7px;
    background-color: #ff3366;
    left: 0px;
    right: 0px;
    top: 0px;
    z-index: 2;
  }
</style>


<div id="indication">&nbsp;</div>
Scroll down!!!
<div class="app" style="position: absolute; margin-top: 3000px;">
Boom!</div>
```



Operators Used

- `fromEvent`
- `tap`
- `throttleTime`


HTTP Polling

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Examples

Example 1

By [@barryrowe](#)

This recipe demonstrates one way you can achieve polling an HTTP endpoint on an interval. This is a common task in web applications, and one that RxJS tends to handle really well as the continuous series of HTTP requests and responses is easy to reason about as a stream of data.

([StackBlitz](#))

RxJS Polling Recipe

Select Cats or Meats and start polling! The Cats option will request random [place kitten](#) images, and the meats option will request random blocks of [bacon-ipsum](#).

Polling Status: Started

Cats ☒ Meats ☐

Start polling

Stop polling



```
// Import stylesheets
import './style.css';

import { Observable, Subscription, of, fromEvent, from, empty, merge, timer } from 'rxjs';
import { map, mapTo, switchMap, tap, mergeMap, takeUntil, filter, finalize } from 'rxjs/operators';

declare type RequestCategory = 'cats' | 'meats';

// Constants for Cat Requests
const CATS_URL = "https://placekitten.com/g/{w}/{h}";
function mapCats(response): Observable<string> {

  return from(new Promise((resolve, reject) => {
    var blob = new Blob([response], {type: "image/png"});
    let reader = new FileReader();
    reader.onload = (data: any) => {
      resolve(data.target.result);
    };
    reader.readAsDataURL(blob);
  }));
}

// Constants for Meat Requests
```

```
const MEATS_URL = "https://baconipsum.com/api/?type=meat-and-fil
ler";
function mapMeats(response): Observable<string> {
  const parsedData = JSON.parse(response);
  return of(parsedData ? parsedData[0] : '');
}

/*****
 * Our Operating State
 *****/
// Which type of data we are requesting
let requestCategory: RequestCategory = 'cats';
// Current Polling Subscription
let pollingSub: Subscription;
/*****/

/**
 * This function will make an AJAX request to the given Url, map
 the
 * JSON parsed repsonse with the provided mapper function, and e
mit
 * the result onto the returned observable.
 */
function requestData(url: string, mapFunc: (any) => Observable<s
tring>): Observable<string> {
  console.log(url)
  const xhr = new XMLHttpRequest();
  return from(new Promise<string>((resolve, reject) => {

    // This is generating a random size for a placekitten image
    // so that we get new cats each request.
    const w = Math.round(Math.random() * 400);
    const h = Math.round(Math.random() * 400);
    const targetUrl = url
      .replace('{w}', w.toString())
      .replace('{h}', h.toString());

    xhr.addEventListener("load", () => {
      resolve(xhr.response);
    });
    xhr.open("GET", targetUrl);
    if(requestCategory === 'cats') {
```

```
// Our cats urls return binary payloads
// so we need to respond as such.
xhr.responseType = "arraybuffer";
}
xhr.send();
}))
.pipe(
  switchMap((data) => mapFunc(xhr.response)),
  tap((data) => console.log('Request result: ', data))
);
}

/**
 * This function will begin our polling for the given state, and
 * on the provided interval (defaulting to 5 seconds)
 */
function startPolling(category: RequestCategory, interval: number = 5000): Observable<string> {
  const url = category === 'cats' ? CATS_URL : MEATS_URL;
  const mapper = category === 'cats' ? mapCats : mapMeats;

  return timer(0, interval)
    .pipe(
      switchMap(_ => requestData(url, mapper))
    );
}

// Gather our DOM Elements to wire up events
const startButton = document.getElementById('start');
const stopButton = document.getElementById('stop');
const text = document.getElementById('text');
const pollingStatus = document.getElementById('polling-status');
const catsRadio = document.getElementById('catsCheckbox');
const meatsRadio = document.getElementById('meatsCheckbox');
const catsClick$ = fromEvent(catsRadio, 'click').pipe(mapTo('cats'));
const meatsClick$ = fromEvent(meatsRadio, 'click').pipe(mapTo('meats'));
const catImage: HTMLImageElement = <HTMLImageElement>document.getElementById('cat');
// Stop polling
```

```
let stopPolling$ = fromEvent(stopButton, 'click');

function updateDom(result) {
  if (requestCategory === 'cats') {
    catImage.src = result;
    console.log(catImage);
  } else {
    text.innerHTML = result;
  }
}

function watchForData(category: RequestCategory) {
  // Start new Poll
  return startPolling(category, 5000).pipe(
    tap(updateDom),
    takeUntil(
      // stop polling on either button click or change of categories
      merge(
        stopPolling$,
        merge(catsClick$, meatsClick$).pipe(filter(c => c !== category))
      )
    ),
    // for demo purposes only
    finalize(() => pollingStatus.innerHTML = 'Stopped')
  )
}

// Handle Form Updates
catsClick$
  .subscribe((category: RequestCategory) => {
    requestCategory = category;
    catImage.style.display = 'block';
    text.style.display = 'none';
  });

meatsClick$
  .subscribe((category: RequestCategory) => {
    requestCategory = category;
    catImage.style.display = 'none';
    text.style.display = 'block';
  });
```

```
});

// Start Polling
fromEvent(startButton, 'click')
  .pipe(
    // for demo purposes only
    tap(_ => pollingStatus.innerHTML = 'Started'),
    mergeMap(_ => watchForData(requestCategory))
  )
  .subscribe();
```

Operators Used

- [filter](#)
- [fromEvent](#)
- [from](#)
- [map](#)
- [mapTo](#)
- [merge](#)
- [mergeMap](#)
- [switchMap](#)
- [timer](#)

Example 2: Simple http polling

By [@adamlubek](#)

This recipe demonstrates polling an HTTP endpoint using `repeat`. It waits for 3 seconds following the response to poll again. Code below is simplified to demonstrate bare bones of solution but link below contains verbose logging and error handling.

([StackBlitz](#))

```
// RxJS v6+
import { of } from 'rxjs';
import { delay, tap, mergeMap, repeat } from 'rxjs/operators';

const fakeDelayedRequest = () => of(new Date()).pipe(delay(1000))
;

const display = response => {
  document.open();
  document.write(response);
};

const poll = of({}).pipe(
  mergeMap(_ => fakeDelayedRequest()),
  tap(display),
  delay(3000),
  repeat()
);


poll.subscribe();
```



Lockscreen

By [adamlubek](#)


This recipe demonstrates RxJS implementation of lockscreen functionality (known for example from smartphones).

 Ultimate Courses™

Fast-track your **RxJS** skills to build **any project** or **get hired** at a top company.

Change your life and career - we will get you there!

Show me how! →



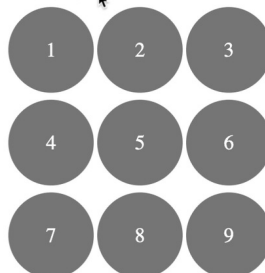
with Brian Troncone
Creator of **LearnRxJS.io**

Example Code

([StackBlitz](#))

Expected Password:

Password: 



- Use mouse to 'swipe' across the lock pad (hold mouse button and swipe :)).
- Pad will turn green if password is correct or red if password is incorrect.
- You can set password to whatever sequence you like.

index.ts

```
/*
  Use mouse to 'swipe' across the lock pad (hold mouse button and
  swipe :).
  Pad will turn green if password is correct or red if password
  is incorrect.
  You can set password to whatever sequence you like.
*/
// RxJS v6+
import { from, fromEvent, Subject, merge, pipe } from 'rxjs';
import {
  switchMap,
  takeUntil,
  repeat,
  tap,
  map,
  throttleTime,
  distinctUntilChanged,
  filter,
  toArray,
  sequenceEqual,
  pluck
} from 'rxjs/operators';
import {
  displaySelectedNumbersSoFar,
  markTouchedPad,
  pads,
  resetPasswordPad,
  setResult
} from './dom-updater';

const sub = new Subject();
const expectedPasswordUpdate$ = fromEvent(
  document.getElementById('expectedPassword'),
  'keyup'
).pipe(
  map((e: any) => e.target.value),
  tap(pass => sub.next(pass.split('').map(e => parseInt(e))))
);
let expectedPassword = [1, 2, 5, 2];
```



```
const expectedPassword$ = sub.pipe(tap((v: any) => (expectedPassword = v)));

const takeMouseSwipe = pipe(
  // take mouse moves
  switchMap(_ => fromEvent(document, 'mousemove')),
  // once mouse is up, we end swipe
  takeUntil(fromEvent(document, 'mouseup')),
  throttleTime(50)
);

const checkIfPasswordMatch = password =>
  from(password).pipe(sequenceEqual(from(expectedPassword)));
const getXCoordsOfMousePosition = ({ clientX, clientY }: MouseEvent) => ({
  x: clientX,
  y: clientY
});
const findSelectedPad = v =>
  pads.find(
    r => v.x > r.left && v.x < r.right && v.y > r.top && v.y < r.bottom
  );
const getIdOfSelectedPad = pipe(
  filter(v => !!v),
  pluck('id'),
  distinctUntilChanged()
);

const actualPassword$ = fromEvent(document, 'mousedown').pipe(
  // new stream so reset password pad and take swipe until mouse up
  tap(resetPasswordPad),
  takeMouseSwipe,
  // as we swipe, we mark pads as touched and display selected numbers
  map(getXCoordsOfMousePosition),
  map(findSelectedPad),
  getIdOfSelectedPad,
  tap(markTouchedPad),
  tap(displaySelectedNumbersSoFar),
  // we need an array of numbers from current swipe which we can pass to checkIfPasswordMatch
```

```

    toArray(),
    // on mouse up (swipe end), switchMap to new stream to check i
    f password match
    switchMap(checkIfPasswordMatch),
    tap(setResult),
    // takeUntil inside takeMouseSwipe terminated stream so we rep
    eat from beginning (mousedown)
    repeat()
  );

merge(expectedPassword$, expectedPasswordUpdate$, actualPassword$
).subscribe();

```

dom-updater.ts

```

const createPadObject = (id, rectangle) => ({
  id: id,
  left: rectangle.left,
  right: rectangle.right,
  top: rectangle.top,
  bottom: rectangle.bottom
});

const setResultText = text =>
  (document.getElementById('result').innerText = text);

const setPasswordPads = color =>
  Array.from(document.querySelectorAll('.cell')).forEach(
    (v: HTMLElement) => (v.style.background = color)
  );

const getPad = id => document.getElementById(`c${id}`);

export const pads = Array.from({ length: 9 }, (_, n) => n + 1).m
  ap(v =>
    createPadObject(v, getPad(v).getBoundingClientRect())
  );

export const markTouchedPad = v => {
  const pad = getPad(v);

```

```
pad.style.background = 'lightgrey';
if (!pad.animate) return; //animate does not work in IE
const animation: any = [
  { transform: 'scale(0.9)' },
  { transform: 'scale(1)' }
];
const animationOptions = {
  duration: 300,
  iterations: 1
};
pad.animate(animation, animationOptions);
document.getSelection().removeAllRanges();
};

export const setResult = result => {
  setPasswordPads(result ? 'MediumSeaGreen' : 'IndianRed');
  setResultText('Password ' + (result ? 'matches :)' : 'does not
  match :('));
};

export const displaySelectedNumbersSoFar = v =>
  (document.getElementById('result').textContent += v);

export const resetPasswordPad = () => {
  setResultText('');
  setPasswordPads('gray');
};
```

html

```
<style>
  .grid {
    border-spacing: 2px;
  }
  .cell {
    width: 50px;
    height: 50px;
    background: grey;
    display: table-cell;
    border-radius: 50%;
    transform: scale(0.5);
```

```
    text-align: center;
    vertical-align: middle;
    color: white;
}

.pulse div {
    animation-name: pulse;
    animation-duration: 2s;
    animation-iteration-count: infinite;
}

@keyframes pulse {
    from { transform: scale(1); }
    50% { transform: scale(0.99); }
    to { transform: scale(1); }
}
</style>
```

Expected Password:

```
<input id="expectedPassword" value="1252"/>
<hr/>
```

Password:

```
<div class="grid pulse">
  <div>
    <div class="cell" id="c1">1</div>
    <div class="cell" id="c2">2</div>
    <div class="cell" id="c3">3</div>
  </div>
  <div>
    <div class="cell" id="c4">4</div>
    <div class="cell" id="c5">5</div>
    <div class="cell" id="c6">6</div>
  </div>
  <div>
    <div class="cell" id="c7">7</div>
    <div class="cell" id="c8">8</div>
    <div class="cell" id="c9">9</div>
  </div>
</div>

<div id="result"></div>
```


Operators Used

- `distinctUntilChanged`
- `filter`
- `from`
- `fromEvent`
- `map`
- `merge`
- `pluck`
- `repeat`
- `sequenceEqual`
- `Subject`
- `switchMap`
- `takeUntil`
- `tap`
- `throttleTime`
- `toArray`

Matrix Digital Rain

By [adamlubek](#)

This recipe demonstrates RxJS implementation of Matrix Digital Rain.

 Ultimate Courses™

Fast-track your **RxJS** skills to build **any project** or **get hired** at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))



index.ts

```
// RxJS v6+
import { interval } from 'rxjs';
import { scan } from 'rxjs/operators';
import { render } from './html-renderer';
import { markForRemoval, updateDrops, updateMatrix } from './matrix';

interval(300)
  .pipe(
    scan<number, any[]>(matrix => (
      markForRemoval(matrix),
      updateDrops(matrix),
      updateMatrix(matrix)
    ), [])
  ).subscribe(render);
```

matrix.ts

```
const drop = (x: number, y: number) => ({ x, y, d: [], remove: false });
const random = (max: number) => Math.floor(Math.random() * Math.floor(max));
const ranodmChar = () => String.fromCharCode(random(128));

export const markForRemoval = matrix =>
  matrix.forEach(
    drop => (drop.remove = drop.remove ? true : drop.d.length > 20)
  );
export const updateDrops = matrix =>
  matrix.forEach(
    drop =>
      (drop.d = drop.remove
        ? drop.d.slice(1).map(e => ranodmChar())
        : [ranodmChar(), ...drop.d.map(e => ranodmChar())])
  );
export const updateMatrix = matrix => [
  ...matrix,
  drop(random(window.innerHeight) / 4, random(window.innerWidth))
];
```

html-renderer.ts


```
const createElem = drop => {
  const elem = document.createElement('div');
  elem.style.position = 'absolute';
  elem.style.marginTop = drop.x + 'px';
  elem.style.marginLeft = drop.y + 'px';
  elem.style.fontSize = '12px';
  elem.innerHTML = drop.d.reduce((acc, c) => (acc += '<br/>' + c)
, '');
  elem.style['color'] = `rgb(21, ${100 + drop.d.length * 10}, 21)`;
  return elem;
};

export const render = matrix => {
  document.body.innerHTML = '';
  const container = document.createElement('div');
  container.style.position = 'relative';
  matrix.forEach(m => container.appendChild(createElem(m)));
  document.body.appendChild(container);
};
```


Operators Used

- [interval](#)
- [scan](#)

Memory Game

By [adamlubek](#)


This recipe demonstrates an RxJS game to train your memory.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

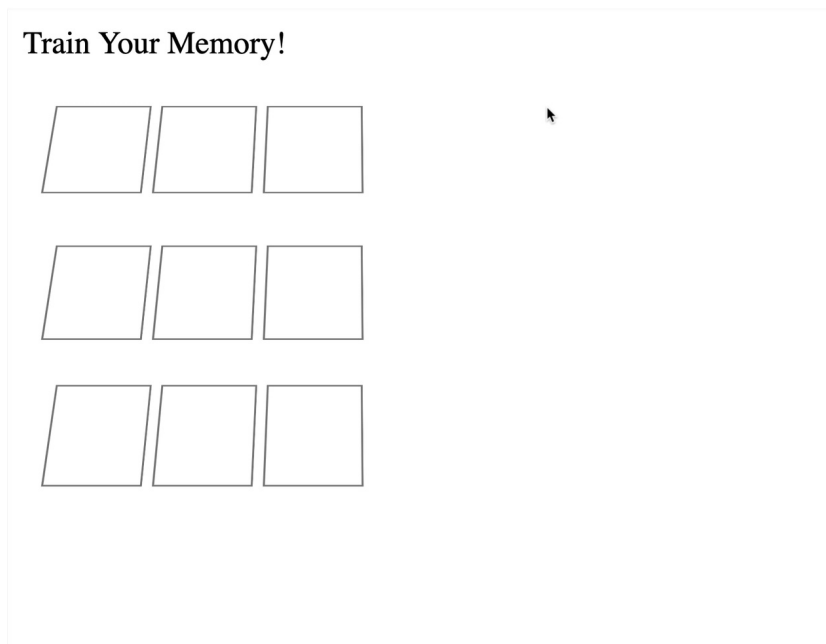
Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))



index.ts

```
// RxJS v6+
import { EMPTY, from, fromEvent, generate, interval, merge, noop
} from 'rxjs';
import {
  map,
  pluck,
  scan,
  sequenceEqual,
  switchMap,
  take,
  tap
} from 'rxjs/operators';

const random = (): number => Math.floor(Math.random() * Math.floor(8));
const setInfo = (text: string) =>
  (document.getElementById('info').innerHTML = text);
const displayLevelChange = () =>
  document
    .querySelectorAll('.child')
    .forEach((c: HTMLElement) => (c.style.background = 'gray'));

const checkIfGameOver$ = (randomSequence: number[]) => (
  userSequence: number[]
) =>
  from(userSequence).pipe(
    sequenceEqual(from(randomSequence)),
    tap(match =>
      !match && userSequence.length === randomSequence.length
        ? setInfo('GAME OVER!')
        : noop
    )
  );

const takePlayerInput$ = (randomSequence: number[]) => _ =>
  fromEvent(document, 'click').pipe(
    take(randomSequence.length),
    scan(
      (acc: number[], curr: MouseEvent) => [
        ...acc,
        parseInt(curr.target['id'])
      ]
    )
  );
```

```

    ],
    []
  ),
  switchMap(checkIfGameOver$(randomSequence)),
  switchMap(result =>
    result
      ? (displayLevelChange(), memoryGame$(randomSequence.length + 1))
      : EMPTY
  )
);

const showSequenceToMemorize$ = (memorySize: number) => (
  randomSequence: number[]
) =>
  interval(1000).pipe(
    tap(i =>
      setInfo(i === memorySize - 1 ? `YOUR TURN` : `${memorySize - i} elements`)
    ),
    take(randomSequence.length),
    map(index => randomSequence[index]),
    tap(value => document.getElementById(`${value}`).click()),
    switchMap(takePlayerInput$(randomSequence))
  );

const memoryGame$ = memorySize =>
  generate(1, x => x <= memorySize, x => x + 1).pipe(
    scan((acc: number[], _: number): number[] => [...acc, random(
) + 1], []),
    switchMap(showSequenceToMemorize$(memorySize))
  );

const elementClick$ = (event: string, color: string) =>
  fromEvent(document.querySelectorAll('.child'), event).pipe(
    pluck('srcElement'),
    tap((e: HTMLElement) => (e.style.background = color))
  );

const clicks$ = merge(
  elementClick$('click', 'lightgray'),
  elementClick$('transitionend', 'white')

```

```
);  
  
const game$ = merge(clicks$, memoryGame$(2));  
  
game$.subscribe();
```

index.html

```
<style>  
  .parent {  
    border-spacing: 5px;  
    width: 50%;  
    padding: 0.5em;  
  }  
  
  .parent.perspective {  
    perspective: 50em;  
  }  
  
  .child {  
    margin: 0.5em;  
    max-width: 2em;  
    min-width: 2em;  
    height: 2.8em;  
    padding: 0.5em;  
    display: table-cell;  
    border: 1px solid rgba(0, 0, 0, 0.5);  
  }  
  
  .parent.perspective .child {  
    transform: rotateX(40deg);  
    transition: all 0.3s ease-in;  
  }  
</style>  
  
<div id="info">Train Your Memory!</div>  
<div id="grid" class="grid parent perspective">  
  <div>  
    <div class="child" id="1"></div>  
    <div class="child" id="2"></div>
```

```
<div class="child" id="3"></div>
</div>
<div>
  <div class="child" id="4"></div>
  <div class="child" id="5"></div>
  <div class="child" id="6"></div>
</div>
<div>
  <div class="child" id="7"></div>
  <div class="child" id="8"></div>
  <div class="child" id="9"></div>
</div>
</div>
```


Operators Used

- empty
- from
- fromEvent
- generate
- interval
- map
- merge
- noop
- pluck
- scan
- sequenceEqual
- switchMap
- take
- tap

Mine Sweeper Game

By [adamlubek](#)


This recipe demonstrates RxJS implementation of Mine Sweeper Game.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



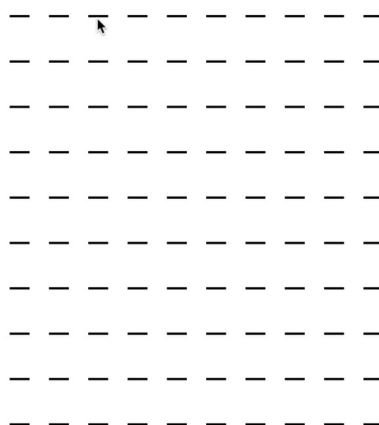
with Brian Troncone

Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))

Score: 0



index.ts

```
// RxJS v6+
import { fromEvent, of } from 'rxjs';
import {
  map,
  tap,
  filter,
  pluck,
  switchMap,
  takeWhile,
  finalize
} from 'rxjs/operators';
import { renderMinefield, renderScore, renderGameOver } from './html-renderer';
import { size, mine } from './constants';
import { addMines, addMarks } from './mines';

const mines$ = of(
  Array(size)
    .fill(0)
    .map(e => Array(size).fill(0))
).pipe(
  map(addMines),
  map(addMarks),
  tap(renderMinefield)
);

const click$ = mines =>
  fromEvent(document, 'click').pipe(
    map(({ clientX, clientY }: MouseEvent) =>
      document.elementFromPoint(clientX, clientY)
    ),
    filter(elem => elem.id !== ''),
    tap(elem =>
      (val => (
        renderScore(val === mine || elem.innerHTML !== '_' ? 0 :
val),
        (elem.innerHTML = val)
      ))(mines[elem.id[0]][elem.id[1]])
    ),
    pluck('id'),
    takeWhile(([x, y]) => mines[x][y] !== mine),
```



```
        finalize(renderGameOver)
    );

    mines$.pipe(switchMap(click$)).subscribe();
```

mines.ts

```
import { size, mine } from './constants';

const randomNumber = () => Math.floor(Math.random() * Math.floor(
size));

export const addMines = arr => {
  for (let i = 0; i < size / 2; i++) {
    arr[randomNumber()][randomNumber()] = mine;
  }

  return arr;
};

const mark = (arr, x, y) =>
  arr[x] !== undefined && arr[x][y] !== undefined
    ? (arr[x][y] += arr[x][y] === mine ? 0 : 1)
    : () => {};

export const addMarks = arr => {
  for (let ri = 0; ri < size; ri++) {
    for (let ci = 0; ci < size; ci++) {
      if (arr[ri][ci] === mine) {
        mark(arr, ri - 1, ci + 1);
        mark(arr, ri - 1, ci);
        mark(arr, ri - 1, ci - 1);
        mark(arr, ri, ci + 1);
        mark(arr, ri, ci - 1);
        mark(arr, ri + 1, ci + 1);
        mark(arr, ri + 1, ci);
        mark(arr, ri + 1, ci - 1);
      }
    }
  }
  return arr;
};
```

constants.ts

```
export const mine = 9;
export const size = 10;
```

html-renderer.ts

```
export const renderMinefield = arr =>
  arr.forEach((r, ri) =>
    (elem =>
      r.forEach(
        (c, ci) =>
          (col => (
            (col.innerText = '_'),
            (col.id = `${ri}${ci}`),
            elem.appendChild(document.createTextNode('\u00A0\u00
A0')),
            elem.appendChild(col)
          ))(document.createElement('span')),
          document.body.appendChild(elem)
        ))(document.createElement('div'))
    );

export const renderScore = val =>
  (scoreElem => (scoreElem.innerText = parseInt(scoreElem.innert
ext) + val))(
    document.getElementById('score')
  );

export const renderGameOver = () =>
  (document.body.innerHTML += '<br/>GAME OVER');

const addElem = decorator =>
  (elem => (decorator(elem), document.body.appendChild(elem)))(
    document.createElement('span')
  );

addElem(elem => (elem.innerText = 'Score: '));
addElem(elem => ((elem.id = 'score'), (elem.innerText = '0')));
```

Operators Used

- [filter](#)
- [finalize](#)
- [fromEvent](#)
- [map](#)
- [of](#)
- [pluck](#)
- [switchMap](#)
- [takeWhile](#)
- [tap](#)

Platform Jumper Game

By *adamlubek*


This recipe demonstrates RxJS implementation of Platform Jumper game.

U Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of **LearnRxJS.io**

Example Code

(StackBlitz)

Lives: 3 Score: 0

[illegible]

index.ts

```
// RxJS v6+
import { interval, of, fromEvent, combineLatest } from 'rxjs';
import { scan, tap, startWith, pluck, switchMap, takeWhile } from
  'rxjs/operators';
import { gameSize } from './constants';
import { render } from './html-renderer';
import { Player, Platform } from './interfaces';
import { updatePlayer, updatePlatforms, initialPlatforms, initialPlayer, handleCollisions, handleKeypresses } from './game';

const gameSpeed = 500;

const platforms$ = interval(gameSpeed)
  .pipe(
    scan<number, Platform[]>(updatePlatforms, initialPlatforms)
  );

const keys$ = (initialPlayer: Player) => fromEvent(document, 'keydown')
  .pipe(
    startWith({ key: '' }),
    pluck('key'),
    scan<string, Player>(
      (plyr: Player, key: string) => handleKeypresses(plyr, key),
      initialPlayer
    )
  );

const player$ = of(initialPlayer())
  .pipe(
    switchMap(p => combineLatest(interval(gameSpeed / 4), keys$(p)))
  )
  .pipe(
    scan<[number, Player], Player>((_, [__, player]) => updatePlayer(player))
  );

combineLatest(player$, platforms$)
  .pipe(
    tap(
      ([player, platforms]) => {
        render(player, platforms);
      }
    )
  );
```

```

    scan<[Player, Platform[]], [Player, Platform[]]>(
      (_, [player, platforms]) => handleCollisions([player, plat
forms])),
    tap(render),
    takeWhile(([player, platforms]) => player.lives > 0)
  )
  .subscribe()

```

game.ts

```

import { Player, Platform } from './interfaces';
import { gameSize } from './constants';

const newPlatform = (x, y): Platform => ({ x, y, scored: false })
;
const newPlayer = (x, y, jumpValue, score, lives): Player => ({
  x,
  y,
  jumpValue,
  canJump: false,
  score: score,
  lives: lives
});
const startingY = 4;
export const initialPlayer = (): Player => newPlayer(0, startingY
, 0, 0, 3);
export const initialPlatforms = [newPlatform(gameSize / 2, start
ingY)];

const random = y => {
  let min = Math.ceil(y - 4);
  let max = Math.floor(y + 4);
  min = min < 0 ? 0 : min;
  max = max > gameSize - 1 ? gameSize - 1 : max;

  return Math.floor(Math.random() * (max - min + 1)) + min;
};

export const updatePlatforms = (platforms: Platform[]): Platform[
] => (

```

```
platforms[platforms.length - 1].x > gameSize / 5
  ? platforms.push(newPlatform(1, random(platforms[platforms.l
length - 1].y)))
  : () => {},
platforms.filter(e => e.x < gameSize - 1).map(e => newPlatform(
e.x + 1, e.y))
);

export const handleKeypresses = (player: Player, key: string) =>
  key === 'ArrowRight'
    ? newPlayer(
      player.x,
      player.y + (player.y < gameSize - 1 ? 1 : 0),
      player.jumpValue,
      player.score,
      player.lives
    )
    : key === 'ArrowLeft'
      ? newPlayer(
        player.x,
        player.y - (player.y > 0 ? 1 : 0),
        player.jumpValue,
        player.score,
        player.lives
      )
      : key === 'ArrowUp'
        ? newPlayer(
          player.x,
          player.y,
          player.x === gameSize - 1 || player.canJump ? 6 : 0,
          player.score,
          player.lives
        )
        : player;

export const updatePlayer = (player: Player): Player => (
  (player.jumpValue -= player.jumpValue > 0 ? 1 : 0),
  (player.x -= player.x - 3 > 0 ? player.jumpValue : 0),
  (player.x += player.x < gameSize - 1 ? 1 : 0),
  player.x === gameSize - 1 ? ((player.lives -= 1), (player.x = 1
)) : () => {},
  player
```



```
);

const handleCollidingPlatform = (
  collidingPlatform: Platform,
  player: Player
) => {
  if (player.canJump) {
    return;
  }


  if (!collidingPlatform) {
    player.canJump = false;
    return;
  }

  if (!collidingPlatform.scored) {
    player.score += 1;
  }
  collidingPlatform.scored = true;

  player.canJump = true;
};

export const handleCollisions = ([player, platforms]: [Player, Platform[]]): [
  Player,
  Platform[]
] => (
  handleCollidingPlatform(
    platforms.find(p => p.x - 1 === player.x && p.y === player.y)
  ,
    player
  ),
  (player.x = player.canJump
    ? (collidingPlatforms =>
      collidingPlatforms.length
        ? (platform => platform.x - 1)(
          collidingPlatforms[collidingPlatforms.length - 1]
        )
      : player.x)(
    platforms.filter(p => p.y === player.y && p.x >= player.x
  )
)
```

```
    )  
    : player.x),  
    [player, platforms]  
  );
```



interfaces.ts

```
export interface Player {  
  x: number;  
  y: number;  
  jumpValue: number;  
  canJump: boolean;  
  score: number;  
  lives: number;  
}  
  
export interface Platform {  
  x: number;  
  y: number;  
  scored: boolean;  
}
```

constants.ts

```
export const gameSize = 20;
```

html-renderer.ts

```
import { gameSize } from './constants';
import { Player, Platform } from './interfaces';

export const render = ([player, platforms]: [Player, Platform[]])
=> {
  document.body.innerHTML = `Lives: ${player.lives} Score: ${pla
yer.score} </br>`;

  const game = Array(gameSize)
    .fill(0)
    .map(_ => Array(gameSize).fill(0));
  game[player.x][player.y] = '*';
  platforms.forEach(p => (game[p.x][p.y] = '_'));

  game.forEach(r => {
    r.forEach(c => (document.body.innerHTML += c === 0 ? '...' :
c));
    document.body.innerHTML += '<br/>';
  });
};
```

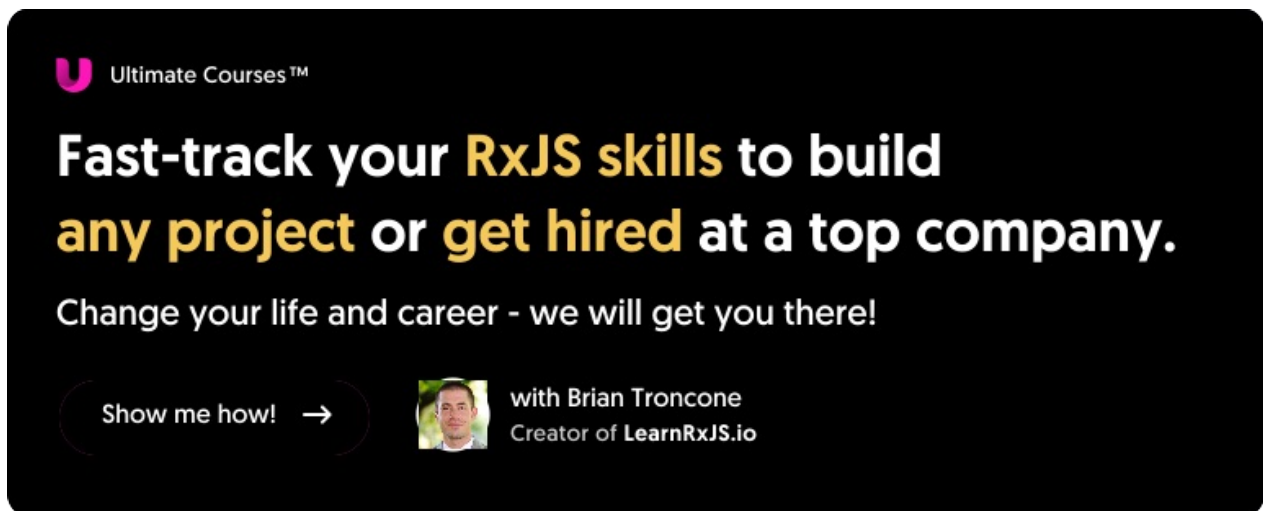
Operators Used

- `combineLatest`
- `fromEvent`
- `interval`
- `of`
- `pluck`
- `scan`
- `startWith`
- `switchMap`
- `takeWhile`
- `tap`

Progress Bar

By [@barryrowe](#)

This recipe demonstrates the creation of an animated progress bar, simulating the management of multiple requests, and updating overall progress as each completes.




U Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

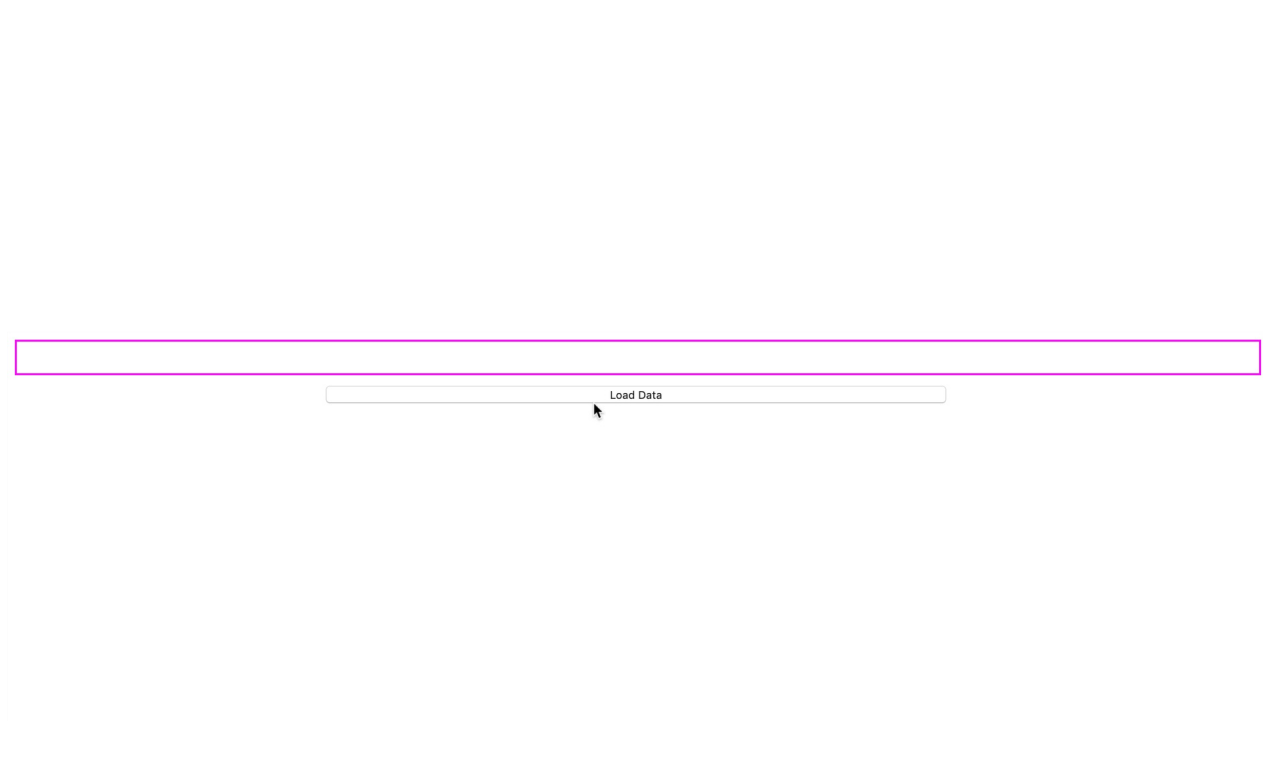
Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))



```
import './style.css';

import { Observable, of, empty, fromEvent, from } from 'rxjs';
import {
  delay,
  switchMapTo,
  concatAll,
  count,
  scan,
  withLatestFrom,
  share
} from 'rxjs/operators';

const requestOne = of('first').pipe(delay(500));
const requestTwo = of('second').pipe(delay(800));
const requestThree = of('third').pipe(delay(1100));
const requestFour = of('fourth').pipe(delay(1400));
const requestFive = of('fifth').pipe(delay(1700));

const loadButton = document.getElementById('load');
const progressBar = document.getElementById('progress');
const content = document.getElementById('data');

// update progress bar as requests complete
const updateProgress = progressRatio => {
  console.log('Progress Ratio: ', progressRatio);
  progressBar.style.width = 100 * progressRatio + '%';
  if (progressRatio === 1) {
    progressBar.className += ' finished';
  } else {
    progressBar.className = progressBar.className.replace(' finished', '');
  }
};

// simple helper to log updates
const updateContent = newContent => {
  content.innerHTML += newContent;
};

const displayData = data => {
  updateContent(`<div class="content-item">${data}</div>`);
};
```

```
};

// simulate 5 separate requests that complete at variable length
const observables: Array<Observable<string>> = [
  requestOne,
  requestTwo,
  requestThree,
  requestFour,
  requestFive
];

const array$ = from(observables);
const requests$ = array$.pipe(concatAll());
const clicks$ = fromEvent(loadButton, 'click');

const progress$ = clicks$.pipe(
  switchMapTo(requests$),
  share()
);

const count$ = array$.pipe(count());

const ratio$ = progress$.pipe(
  scan(current => current + 1, 0),
  withLatestFrom(count$, (current, count) => current / count)
);

clicks$.pipe(switchMapTo(ratio$)).subscribe(updateProgress);

progress$.subscribe(displayData);
```

html

```
<div class="progress-container">
  <div class="progress" id="progress"></div>
</div>

<button id="load">
  Load Data
</button>

<div id="data"></div>
```


Thanks to [@johnlinquist](#) for the additional help with example!

Operators Used

- [concatAll](#)
- [delay](#)
- [fromEvent](#)
- [from](#)
- [scan](#)
- [share](#)
- [switchMap](#)
- [withLatestFrom](#)

Save Indicator


This recipe demonstrates the creation of a google docs-esque save indicator with RxJS.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of LearnRxJS.io

Example Code

([StackBlitz](#))

Take a note!

All changes saved

```
import { fromEvent, of, merge, empty, concat, defer } from 'rxjs'
;
```



```
import {
  delay,
  map,
  mergeMap,
  tap,
  debounceTime,
  distinctUntilChanged,
  mapTo,
  filter,
  share,
  switchAll
} from 'rxjs/operators';
import { format } from 'date-fns';

// track in progress saves
let savesInProgress = 0;

// references
const input = document.getElementById('note-input');
const saveIndicator = document.querySelector('.save-indicator');

// streams
const keyup$ = fromEvent(input, 'keyup');

// fake save request
const saveChanges = value => {
  return of(value).pipe(delay(1500));
};


/**
 * Trigger a save when the user stops typing for 200ms
 * After new data has been successfully saved, so a saved
 * and last updated indicator.
 */
const inputToSave$ = keyup$.pipe(
  debounceTime(200),
  map(e => e.target.value),
  distinctUntilChanged(),
  share()
);

const savesInProgress$ = inputToSave$.pipe(
```

```
mapTo(of('Saving')),  
tap(_ => savesInProgress++)  
);  
  
const savesCompleted$ = inputToSave$.pipe(  
  mergeMap(saveChanges),  
  tap(_ => savesInProgress--),  
  // ignore if additional saves are in progress  
  filter(_ => !savesInProgress),  
  mapTo(  
    concat(  
      // display saved for 2s  
      of('Saved!'),  
      empty().pipe(delay(2000)),  
      // then last updated time, defer for proper time  
      defer(() => of(`Last updated: ${format(Date.now(), 'MM/DD/YYYY hh:mm')}`))  
    )  
  )  
);  
  
merge(savesInProgress$, savesCompleted$)  
  .pipe(  
    /*  
     If new save comes in when our completion observable is running,  
     we want to switch to it for a status update.  
    */  
    switchAll()  
  )  
  .subscribe(status => {  
    saveIndicator.innerHTML = status;  
  });
```

Smart Counter


An interesting element on interfaces which involve dynamically updating numbers is a smart counter, or odometer effect. Instead of jumping a number up and down, quickly counting to the desired number can achieve a cool effect. An example of a popular library that accomplishes this is [odometer](#) by [Hubspot](#). Let's see how we can accomplish something similar with just a few lines of RxJS.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

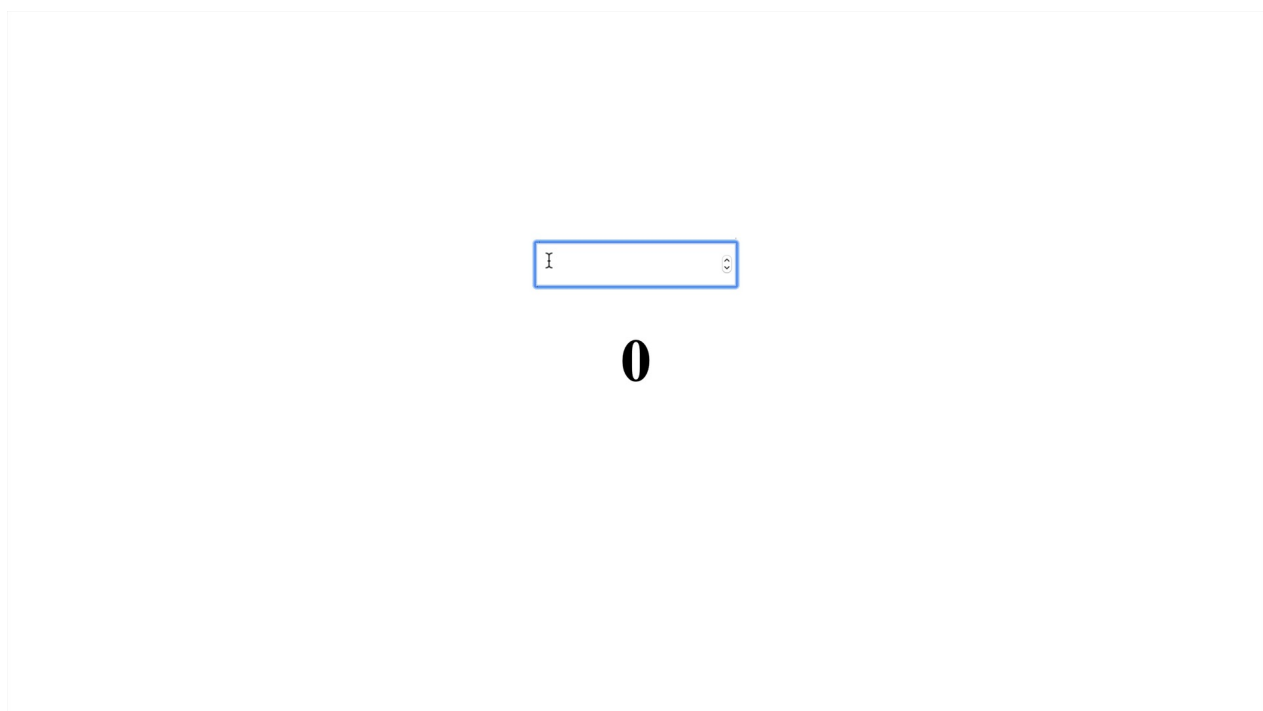
Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Vanilla JS

([StackBlitz](#))



```
import { timer, fromEvent, merge } from 'rxjs';
import {
  switchMap,
  startWith,
  scan,
  takeWhile,
  takeUntil,
  filter,
  mapTo,
  map,
  tap,
  pluck
} from 'rxjs/operators';

let currentNumber = 0;

// elems
const input: any = document.getElementById('range');

// utility functions
const takeUntilFunc = (endRange, currentNumber) => {
  return endRange > currentNumber
    ? val => val <= endRange
    : val => val >= endRange;
};

const positiveOrNegative = (endRange, currentNumber) => {
  return endRange > currentNumber ? 1 : -1;
};

const updateHTML = id => val => (document.getElementById(id).innerHTML = val);

// streams
const enter$ = fromEvent(input, 'keyup').pipe(
  pluck('code'),
  filter(code => code === 'Enter')
);

enter$
  .pipe(
    map(() => parseInt(input.value)),
    switchMap(endRange => {
```

```
return timer(0, 20).pipe(
  mapTo(positiveOrNegative(endRange, currentNumber)),
  startWith(currentNumber),
  scan((acc, curr) => acc + curr),
  takeWhile(takeUntilFunc(endRange, currentNumber))
);
}),
tap(v => (currentNumber = v)),
startWith(currentNumber)
)
.subscribe(updateHTML('display'));
```

HTML

```
<div class="container">
  <input id="range" type="number" />
  <h1 id="display"></h1>
</div>
```

We can easily take our vanilla smart counter and wrap it in any popular component based UI library. Below is an example of an Angular smart counter component which takes an `Input` of the updated end ranges and performs the appropriate transition.

Angular Version

([StackBlitz](#))

```
import { Component, Input, OnDestroy } from '@angular/core';
import { Subject } from 'rxjs/Subject';
import { timer } from 'rxjs/observable/timer';
import { switchMap, startWith, scan, takeWhile, takeUntil, mapTo
} from 'rxjs/operators';

@Component({
  selector: 'number-tracker',
  template: `
    <h3> {{ currentNumber }}</h3>
  `
})
```

```

}))
export class NumberTrackerComponent implements OnDestroy {
  @Input()
  set end(endRange: number) {
    this._counterSub$.next(endRange);
  }
  @Input() countInterval = 20;
  public currentNumber = 0;
  private _counterSub$ = new Subject();
  private _onDestroy$ = new Subject();

  constructor() {
    this._counterSub$
      .pipe(
        switchMap(endRange => {
          return timer(0, this.countInterval).pipe(
            mapTo(this.positiveOrNegative(endRange, this.current
Number)),
            startWith(this.currentNumber),
            scan((acc: number, curr: number) => acc + curr),
            takeWhile(this.isApproachingRange(endRange, this.cur
rentNumber))
          )
        })
      .takeUntil(this._onDestroy$)
      .subscribe((val: number) => this.currentNumber = val);
  }

  private positiveOrNegative(endRange, currentNumber) {
    return endRange > currentNumber ? 1 : -1;
  }

  private isApproachingRange(endRange, currentNumber) {
    return endRange > currentNumber
      ? val => val <= endRange
      : val => val >= endRange;
  }

  ngOnDestroy() {
    this._onDestroy$.next();
    this._onDestroy$.complete();
  }
}

```

```
}  
}
```

HTML

```
<p>  
  <input  
    type="number"  
    (keyup.enter)="counterNumber = vanillaInput.value"  
    #vanillaInput  
  />  
  <button (click)="counterNumber = vanillaInput.value">  
    Update number  
  </button>  
</p>  
<number-tracker [end]="counterNumber"></number-tracker>
```


Operators Used

- `fromEvent`
- `map`
- `mapTo`
- `scan`
- `startWith`
- `switchMap`
- `takeWhile`

Space Invaders Game

By [adamlubek](#)


This recipe demonstrates RxJS implementation of Space Invaders Game.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone

Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))

Score: 0 Lives: 3



index.ts


```
// RxJS v6+
import { fromEvent, interval } from 'rxjs';
import {
  map,
  scan,
  tap,
  startWith,
  withLatestFrom,
  takeUntil,
  repeat
} from 'rxjs/operators';
import { gameUpdate, initialState } from './game';
import { State, Input } from './interfaces';
import { paint } from './html-renderer';

const spaceInvaders$ = interval(100).pipe(
  withLatestFrom(
    fromEvent(document, 'keydown').pipe(
      startWith({ code: '' }),
      takeUntil(fromEvent(document, 'keyup')),
      repeat()
    )
  ),
  map(([interval, event]: [number, KeyboardEvent]): Input => ({
    delta: interval,
    key: event.code
  })),
  scan(gameUpdate, initialState),
  tap(e => paint(e.game, e.playerLives, e.score, e.isGameOver))
);

spaceInvaders$.subscribe();
```

game.ts

```
import { State, Input } from './interfaces';
import { empty, player, invader, shot, noOfInvadersRows } from './constants';

const gameObject = (x, y) => ({ x: x, y: y });
```

```
const gameSize = 20;
const clearGame = () =>
  Array(gameSize)
    .fill(empty)
    .map(e => Array(gameSize).fill(empty));

const createInvaders = () =>
  Array.from(Array(noOfInvadersRows).keys()).reduce(
    (invds, row) => [...invds, ...createRowOfInvaders(row)],
    []
  );
const createRowOfInvaders = row =>
  Array.from(Array(gameSize / 2).keys())
    .filter(e => (row % 2 === 0 ? e % 2 === 0 : e % 2 !== 0))
    .map(e => gameObject(row, e + 4));

const invadersDirection = (state: State): number =>
  state.invaders.length && state.invaders[0].y <= 0
    ? 1
    : state.invaders.length &&
      state.invaders[state.invaders.length - 1].y >= gameSize - 1
    ? -1
    : state.invadersDirY;

const drawGame = (state: State): number[][] => (
  keepShipWithinGame(state),
  (state.game = clearGame()),
  (state.game[state.game.length - 1][state.shipY] = player),
  state.invaders.forEach(i => (state.game[i.x][i.y] = invader)),
  state.invadersShoots.forEach(s => (state.game[s.x][s.y] = shot))
),
  state.shoots.forEach(s => (state.game[s.x][s.y] = shot)),
  state.game
);

const addInvaderShoot = state =>
  (randomInvader => gameObject(randomInvader.x, randomInvader.y))
  (
    state.invaders[Math.floor(Math.random() * state.invaders.length)]
  );
```

```

const collision = (e1, e2) => e1.x === e2.x && e1.y === e2.y;
const filterOutCollisions = (c1: any[], c2: any[]): any[] =>
  c1.filter(e1 => !c2.find(e2 => collision(e1, e2)));
const updateScore = (state: State): number =>
  state.shoots.find(s => state.invaders.find(i => collision(s, i)
))
  ? state.score + 1
  : state.score;

const updateState = (state: State): State => ({
  delta: state.delta,
  game: drawGame(state),
  shipY: state.shipY,
  playerLives: state.invadersShoots.some(
    e => e.x === gameSize - 1 && e.y === state.shipY
  )
  ? state.playerLives - 1
  : state.playerLives,
  isGameOver: state.playerLives <= 0,
  score: updateScore(state),
  invadersDirY: invadersDirection(state),
  invaders: !state.invaders.length
    ? createInvaders()
    : filterOutCollisions(state.invaders, state.shoots).map(i =>
      state.delta % 10 === 0
        ? gameObject(
            i.x + (state.delta % (state.shootFrequency + 10)) =
            == 0 ? 1 : 0),
            i.y + state.invadersDirY
          )
        : i
    ),
  invadersShoots:
    ((state.invadersShoots =
      state.delta % state.shootFrequency === 0
        ? [...state.invadersShoots, addInvaderShoot(state)]
        : state.invadersShoots),
    state.invadersShoots
      .filter(e => e.x < gameSize - 1)
      .map(e => gameObject(e.x + 1, e.y))),
  shoots: filterOutCollisions(state.shoots, state.invaders)

```

```
.filter(e => e.x > 0)
.map(e => gameObject(e.x - 1, e.y)),
shootFrequency: !state.invaders.length
  ? state.shootFrequency - 5
  : state.shootFrequency
});

const keepShipWithinGame = (state: State): number => (
  (state.shipY = state.shipY < 0 ? 0 : state.shipY),
  (state.shipY = state.shipY >= gameSize - 1 ? gameSize - 1 : state.shipY)
);

const updateShipY = (state: State, input: Input): number =>
  input.key !== 'ArrowLeft' && input.key !== 'ArrowRight'
    ? state.shipY
    : (state.shipY -= input.key === 'ArrowLeft' ? 1 : -1);

const addShots = (state: State, input: Input) =>
  (state.shoots =
    input.key === 'Space'
      ? [...state.shoots, gameObject(gameSize - 2, state.shipY)]
      : state.shoots);

const isGameOver = (state: State): boolean =>
  state.playerLives <= 0 ||
  (state.invaders.length &&
    state.invaders[state.invaders.length - 1].x >= gameSize - 1);

export const initialState: State = {
  delta: 0,
  game: clearGame(),
  shipY: 10,
  playerLives: 3,
  isGameOver: false,
  score: 0,
  invadersDirY: 1,
  invaders: createInvaders(),
  invadersShoots: [],
  shoots: [],
  shootFrequency: 20
}
```

```
};

const processInput = (state: State, input: Input) => (
  updateShipY(state, input), addShots(state, input)
);
const whileNotGameOver = (state: State, input: Input) =>
  (state.delta = isGameOver(state) ? undefined : input.dlta);

export const gameUpdate = (state: State, input: Input): State =>
(
  whileNotGameOver(state, input), processInput(state, input), up
dateState(state)
);
```

constants.ts

```
export const empty = 0;
export const player = 1;
export const invader = 2;
export const shot = 3;
export const noOfInvadersRows = 6;
```

interfaces.ts

```
export interface State {
  delta: number;
  game: number[][];
  shipY: number;
  playerLives: number;
  isGameOver: boolean;
  score: number;
  invadersDirY: number;
  invaders: any[];
  invadersShoots: any[];
  shoots: any[];
  shootFrequency: number;
}

export interface Input {
  dlta: number;
  key: string;
}
```

html-renderer.ts

```
import { empty, player, invader, shot } from './constants';

const createElem = col => {
  const elem = document.createElement('div');
  elem.classList.add('board');
  elem.style.display = 'inline-block';
  elem.style.marginLeft = '10px';
  elem.style.height = '6px';
  elem.style.width = '6px';
  elem.style['background-color'] =
    col === empty
      ? 'white'
      : col === player
        ? 'cornflowerblue'
        : col === invader
          ? 'gray'
          : 'silver';
  elem.style['border-radius'] = '90%';
  return elem;
```

```
};

export const paint = (
  game: number[][],
  playerLives: number,
  score: number,
  isGameOver: boolean
) => {
  document.body.innerHTML = '';
  document.body.innerHTML += `Score: ${score} Lives: ${playerLives}`;

  if (isGameOver) {
    document.body.innerHTML += ' GAME OVER!';
    return;
  }

  game.forEach(row => {
    const rowContainer = document.createElement('div');
    row.forEach(col => rowContainer.appendChild(createElem(col)))
  });
  document.body.appendChild(rowContainer);
};
```


Operators Used

- [fromEvent](#)
- [interval](#)
- [map](#)
- [repeat](#)
- [scan](#)
- [startWith](#)
- [takeUntil](#)
- [tap](#)
- [withLatestFrom](#)

Stop Watch

By [adamlubek](#)


This recipe demonstrates RxJS implementation of Stop Watch, inspired by [RxJS Advanced Patterns – Operate Heavily Dynamic UI's](#) talk by [@Michael_Hladky](#)

 Ultimate Courses™

Fast-track your **RxJS** skills to build **any project** or **get hired** at a top company.

Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of **LearnRxJS.io**

Example Code

([StackBlitz](#))

0

Setup

start pause reset

Count

count up count down

Set to

0

set value

Speed

1000

set speed

Increase

1

set increase

index.ts

```
// RxJS v6+
import { fromEvent, interval, merge, noop, NEVER } from 'rxjs';
import { map, mapTo, scan, startWith, switchMap, tap } from 'rxjs/operators';

interface State {
  count: boolean;
  countup: boolean;
  speed: number;
  value: number;
  increase: number;
}

const getElem = (id: string): HTMLElement => document.getElementById(id);
const getVal = (id: string): number => parseInt((getElem(id))['value']);
const fromClick = (id: string) => fromEvent(getElem(id), 'click');
;
const fromClickAndMapTo = (id: string, obj: {}) => fromClick(id).pipe(mapTo(obj));
const fromClickAndMap = (id: string, fn: (_) => {}) => fromClick(id).pipe(map(fn));
const setValue = (val: number) => getElem('counter').innerText = val.toString()

const events$ =
  merge(
    fromClickAndMapTo('start', { count: true }),
    fromClickAndMapTo('pause', { count: false }),
    fromClickAndMapTo('reset', { value: 0 }),
    fromClickAndMapTo('countup', { countup: true }),
    fromClickAndMapTo('countdown', { countup: false }),
    fromClickAndMap('setto', _ => ({ value: getVal('value') })),
    fromClickAndMap('setspeed', _ => ({ speed: getVal('speed') })
  ),
    fromClickAndMap('setincrease', _ => ({ increase: getVal('increase') })))
  );
```

```
const stopWatch$ = events$.pipe(
  startWith({ count: false, speed: 1000, value: 0, countup: true,
    increase: 1 }),
  scan((state: State, curr): State => ({ ...state, ...curr }), {}
),
  tap((state: State) => setValue(state.value)),
  switchMap((state: State) => state.count
    ? interval(state.speed)
      .pipe(
        tap(_ => state.value += state.countup ? state.increase :
- state.increase),
        tap(_ => setValue(state.value))
      )
    : NEVER)
);

stopWatch$.subscribe();
```

index.html

```
<style>
  input,
  #counter,
  #controls {
    text-align: center;
    margin: auto;
  }

  #counter {
    font-size: 50px;
  }

  #controls {
    width: 50%;
  }
</style>

<div id="counter">0</div>
<div id="controls">
```

```
<fieldset>
  <legend>Setup</legend>
  <button id="start">start</button>
  <button id="pause">pause</button>
  <button id="reset">reset</button>
</fieldset>
<fieldset>
  <legend>Count</legend>
  <button id="countup">count up</button>
  <button id="countdown">count down</button>
</fieldset>
<fieldset>
  <legend>Set to</legend>
  <input id="value" value="0"></input>
  <br/>
  <button id="setto">set value</button>
</fieldset>
<fieldset>
  <legend>Speed</legend>
  <input id="speed" value="1000"></input>
  <br/>
  <button id="setspeed">set speed</button>
</fieldset>
<fieldset>
  <legend>Increase</legend>
  <input id="increase" value="1"></input>
  <br/>
  <button id="setincrease">set increase</button>
</fieldset>
</div>
```

Operators Used

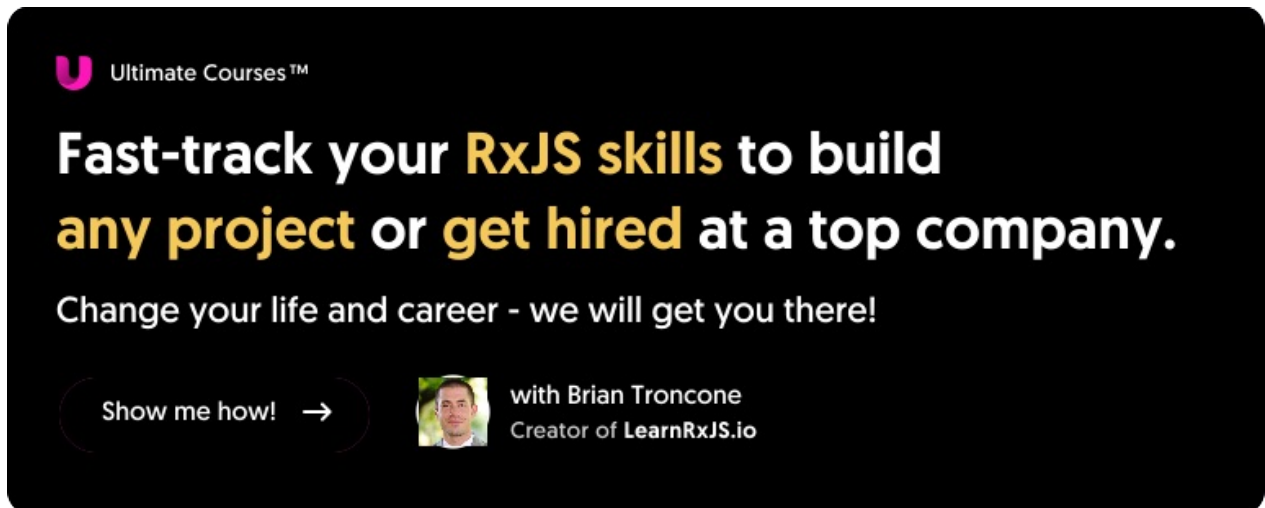
- `fromEvent`
- `interval`
- `map`
- `mapTo`
- `merge`
- `NEVER`

- noop
- scan
- startWith
- switchMap
- tap

Swipe To Refresh

By [adamlubek](#)

This recipe demonstrates RxJS implementation of swipe to refresh functionality.
Inspired by [@BenLesh](#) RxJs talks.




U Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

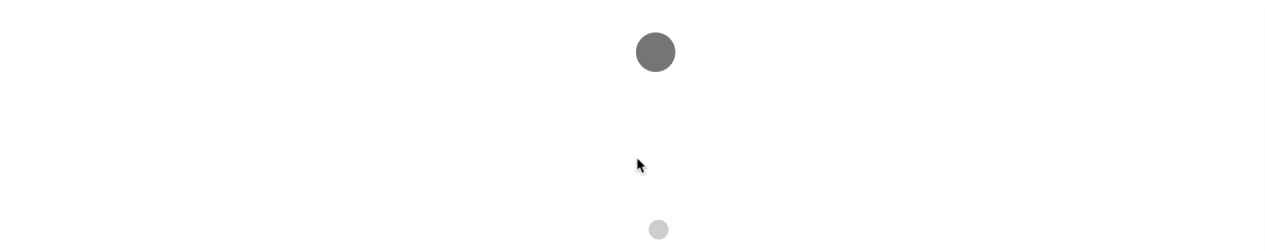
Change your life and career - we will get you there!

Show me how! →

 with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))



Swipe gray dot down to get latest date/time

index.ts

```
// RxJS v6+
import { fromEvent, iif, of, pipe } from 'rxjs';
import {
  finalize,
  mergeMap,
  takeUntil,
  takeWhile,
  repeat,
  map,
  tap,
  exhaustMap,
  delay
} from 'rxjs/operators';

const setRefreshPos = y =>
  (document.getElementById('refresh').style.top = `${y}px`);
const resetRefresh = () => setRefreshPos(10);
const setData = data => (document.getElementById('data').innerText = data);

const fakeRequest = () =>
  of(new Date().toUTCString()).pipe(
    tap(_ => console.log('request')),
    delay(1000)
  );

const takeUntilMouseUpOrRefresh$ = pipe(
  takeUntil(fromEvent(document, 'mouseup')),
  takeWhile(y => y < 110)
);

const moveDot = y => of(y).pipe(tap(setRefreshPos));
const refresh$ = of({}).pipe(
  tap(resetRefresh),
  tap(e => setData('...refreshing...')),
  exhaustMap(_ => fakeRequest()),
  tap(setData)
);

fromEvent(document, 'mousedown')
```

```
.pipe(  
  mergeMap(_ => fromEvent(document, 'mousemove')),  
  map((e: MouseEvent) => e.clientY),  
  takeUntilMouseUpOrRefresh$,  
  finalize(resetRefresh),  
  exhaustMap(y => iif(() => y < 100, moveDot(y), refresh$)),  
  finalize(() => console.log('end')),  
  repeat()  
)  
.subscribe();
```

html


```
<style>
  #refresh {
    position: absolute;
    width: 20px;
    height: 20px;
    background: grey;
    border-radius: 50%;
    left: 50%;
  }

  #point {
    position: absolute;
    width: 10px;
    height: 10px;
    background: lightgrey;
    border-radius: 50%;
    left: 51%;
    top: 105px;
  }

  #data {
    position: absolute;
    top: 150px;
  }
</style>

<div id='refresh'></div>
<div id='point'></div>
<div id='data'>Swipe gray dot down to get latest date/time</div>
```

Operators Used


- [delay](#)
- [exhaustMap](#)
- [finalize](#)
- [fromEvent](#)
- [iif](#)
- [map](#)
- [mergeMap](#)

- of
- repeat
- takeUntil
- takeWhile
- tap

Tank Battle Game

By [adamlubek](#)


This recipe demonstrates RxJS implementation of Tank Battle like game.

 Ultimate Courses™

Fast-track your RxJS skills to build any project or get hired at a top company.

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))

Scores: P1: 0 P1: 0

v

<

This game requires 2 players :)
Player 1 controls: wsad, fire: c
Player 2 controls: ikjl, fire: n

index.ts

```
// RxJS v6+
import { fromEvent, combineLatest, interval } from 'rxjs';
import { scan, tap, startWith } from 'rxjs/operators';
import { gameSize, down, up, right, left, p1Color, p2Color } from
  './constants';
import { State } from './interfaces';
import {
  updatePlayer,
  addShots,
  updateShots,
  checkCollisions,
  initialState
} from './game';
import { paint } from './html-renderer';

combineLatest(
  interval(100),
  fromEvent(document, 'keydown').pipe(startWith({ key: '' }))
)
  .pipe(
    scan < [number, KeyboardEvent],
    State >
    ((state, [_ , event]) => (
      updatePlayer(state.players[0], event.key, 'w', 's', 'a',
'd'),
      updatePlayer(state.players[1], event.key, 'i', 'k', 'j',
'l'),
      addShots(state, event.key),
      (state.shots = updateShots(state.shots)),
      checkCollisions(state),
      state
    ),
    initialState),
    tap(paint)
  )
  .subscribe();
```

game.ts

```
import {
  gameSize,
  down,
  up,
  right,
  left,
  p1Color,
  p2Color,
  p1Shot,
  p2Shot
} from './constants';
import { GameObject, State } from './interfaces';

const gameObject = (x, y, g, c): GameObject => ({ x, y, g, s: 0,
  c: c });
const noop = (): void => {};

export const updatePlayer = (
  p: GameObject,
  key: string,
  u,
  d,
  l,
  r
): GameObject => (
  key === d
    ? ((p.x += p.x < gameSize - 1 ? 1 : 0), (p.g = down))
    : key === u
    ? ((p.x += p.x > 0 ? -1 : 0), (p.g = up))
    : noop,
  key === r
    ? ((p.y += p.y < gameSize - 1 ? 1 : 0), (p.g = right))
    : key === l
    ? ((p.y += p.y > 0 ? -1 : 0), (p.g = left))
    : noop,
  p
);

export const addShot = (player: GameObject): GameObject => ({
  x: player.x,
  y: player.y,
```

```
    g: player.g
  });

export const addShots = (state: State, key: string): void =>
  state.shots.push(
    key === p1Shot
      ? addShot(state.players[0])
      : key === p2Shot
      ? addShot(state.players[1])
      : []
  );

export const updateShots = (shots: GameObject[]): GameObject[] =>

  shots
    .filter(s => s.x > 0 && s.x < gameSize - 1 && s.y > 0 && s.y
< gameSize)
    .map(
      s => (
        s.g === down
          ? (s.x += 1)
          : s.g === up
          ? (s.x += -1)
          : s.g === right
          ? (s.y += 1)
          : s.g === left
          ? (s.y += -1)
          : () => {},
        s
      )
    );

export const initialState: State = {
  players: [
    gameObject(1, 1, right, p1Color),
    gameObject(gameSize - 2, gameSize - 2, left, p2Color)
  ],
  shots: []
};

export const checkCollisions = (state: State): void =>
  state.players.forEach((p, i) => {
```

```
const collidingShotIndex = state.shots.findIndex(
  s => s.x === p.x && s.y === p.y
);
if (collidingShotIndex > -1) {
  if (i === 0) {
    state.players[1].s += 1;
  } else {
    state.players[0].s += 1;
  }
  state.shots.splice(collidingShotIndex, 1);
}
});
```

interfaces.ts

```
export interface GameObject {
  x: number;
  y: number;
  g: string;
  s: number;
  c: string;
}

export interface State {
  players: GameObject[];
  shots: GameObject[];
}
```

constants.ts

```
export const gameSize = 20;
export const up = '^';
export const down = 'v';
export const left = '<';
export const right = '>';
export const empty = 0;
export const p1Color = 'DarkViolet';
export const p2Color = 'CornflowerBlue';
export const p1Shot = 'c';
export const p2Shot = 'n';
```

html-renderer.ts

```
import { gameSize, empty, p1Color, p2Color } from './constants';
import { State, GameObject } from './interfaces';

const createElem = (gameObject: GameObject) => {
  const elem = document.createElement('div');
  elem.style.display = 'inline-block';
  elem.style.marginLeft = '10px';
  elem.style.height = '6px';
  elem.style.width = '6px';
  elem.style.color = gameObject.c;
  elem.innerText = gameObject === empty ? ' ' : gameObject.g;

  return elem;
};

const paintPlayerScore = (score: number, color: string) => {
  const scoreElem = document.createElement('span');
  scoreElem.innerHTML = `P1: ${score}`;
  scoreElem.style.color = color;
  document.body.appendChild(scoreElem);
};

const paintScores = (state: State) => {
  document.body.innerHTML = 'Scores: ';
  paintPlayerScore(state.players[0].s, p1Color);
  paintPlayerScore(state.players[1].s, p2Color);
};
```



```
const painInfo = () => {
  document.body.innerHTML += 'This game requires 2 players :)';
  document.body.innerHTML += '<br/>';
  document.body.innerHTML += 'Player 1 controls: wsad, fire: c';
  document.body.innerHTML += '<br/>';
  document.body.innerHTML += 'Player 2 controls: ikjl, fire: n';
};

const emptyGame = () =>
  Array(gameSize)
    .fill(empty)
    .map(_ => Array(gameSize).fill(empty));
const paintGame = (state: State) => {
  const game = emptyGame();
  state.players.forEach(p => (game[p.x][p.y] = { g: p.g, c: p.c }
));
  state.shots.forEach(s => (game[s.x][s.y] = { g: '*', c: 'black'
})));

  game.forEach(row => {
    const rowContainer = document.createElement('div');
    row.forEach(col => rowContainer.appendChild(createElem(col)))
  });
  document.body.appendChild(rowContainer);
};

export const paint = (state: State) => {
  paintScores(state);
  document.body.innerHTML += '<br/>';
  paintGame(state);
  painInfo();
};
```

Operators Used


- [combineLatest](#)
- [fromEvent](#)

- `interval`
- `scan`
- `startWith`
- `tap`

Tetris Game

By [adamlubek](#)


This recipe demonstrates RxJS implementation of Tetris game.

 Ultimate Courses™

Fast-track your **RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

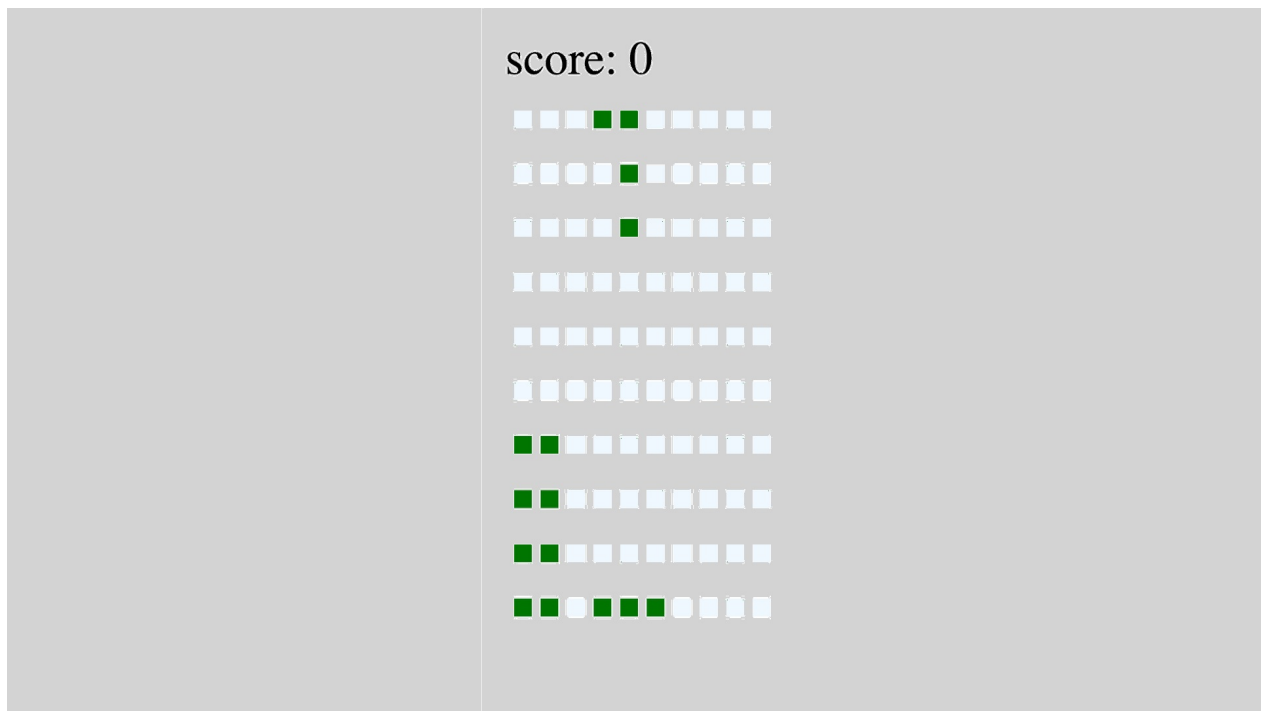
Show me how! →



with Brian Troncone
Creator of **LearnRxJS.io**

Example Code

([StackBlitz](#))



index.ts

```
// RxJS v6+
import { fromEvent, of, interval, combineLatest } from 'rxjs';
import { finalize, map, pluck, scan, startWith, takeWhile, tap }
from 'rxjs/operators';
import { score, randomBrick, clearGame, initialState } from './game';
import { render, renderGameOver } from './html-renderer';
import { handleKeyPress, resetKey } from './keyboard';
import { collide } from './collision';
import { rotate } from './rotation';
import { BRICK } from './constants';
import { State, Brick, Key } from './interfaces';

const player$ = combineLatest(
  of(randomBrick()),
  of({ code: '' }),
  fromEvent(document, 'keyup').pipe(startWith({ code: undefined })),
  pluck('code'))
).pipe(
  map(([brick, key, keyCode]: [Brick, Key, string]) => (key.code =
  keyCode, [brick, key]))
);

const state$ = interval(1000)
  .pipe(
    scan<number, State>((state, _) => (state.x++ , state), initialState)
  );

const game$ = combineLatest(state$, player$)
  .pipe(
    scan<[State, [Brick, Key]], [State, [Brick, Key]]>(
      ([state, [brick, key]]) => (
        state = handleKeyPress(state, brick, key),
        ([newState, rotatedBrick]: [State, Brick]) => (
          state = newState,
          brick = rotatedBrick
        ))(rotate(state, brick, key)),
        ([newState, collidedBrick]: [State, Brick]) => (
          state = newState,
          brick = collidedBrick
        ))
    )
  );
```

```
    ))(collide(state, brick)),  
    state = score(state),  
    resetKey(key),  
    [state, [brick, key]]  
  )),  
  tap([state, [brick, key]]) => render(state, brick),  
  takeWhile([state, [brick, key]]) => !state.game[1].some(c =>  
c === BRICK)),  
  finalize(renderGameOver)  
);  
  
game$.subscribe();
```

game.ts

```
import { GAME_SIZE, EMPTY, BRICK } from './constants';
import { State } from './interfaces';

const bricks = [
  [[0, 0, 0], [1, 1, 1], [0, 0, 0]],
  [[1, 1, 1], [0, 1, 0], [0, 1, 0]],
  [[0, 1, 1], [0, 1, 0], [0, 1, 0]],
  [[1, 1, 0], [0, 1, 0], [0, 1, 0]],
  [[1, 1, 0], [1, 1, 0], [0, 0, 0]]
]

export const clearGame = () => Array(GAME_SIZE).fill(EMPTY).map(e
  => Array(GAME_SIZE).fill(EMPTY));
export const updatePosition = (position: number, column: number)
=> position === 0 ? column : position;
export const validGame = (game: number[][][]) => game.map(r => r.f
  ilter((_, i) => i < GAME_SIZE));
export const validBrick = (brick: number[][][]) => brick.filter(e
  => e.some(b => b === BRICK));
export const randomBrick = () => bricks[Math.floor(Math.random()
  * bricks.length)];

export const score = (state: State): State => (scoreIndex => (
  scoreIndex > -1
    ? (
      state.score += 1,
      state.game.splice(scoreIndex, 1),
      state.game = [Array(GAME_SIZE).fill(EMPTY), ...state.game],

      state
    )
    : state
  ))(state.game.findIndex(e => e.every(e => e === BRICK)));

export const initialState = {
  game: clearGame(),
  x: 0,
  y: 0,
  score: 0
};
```

collision.ts

```

import { GAME_SIZE, BRICK, EMPTY } from './constants';
import { validBrick, validGame, updatePosition, randomBrick } from './game';
import { State, Brick } from './interfaces';

const isGoingToLevelWithExistingBricks = (state: State, brick: Brick): boolean => {
  const gameHeight = state.game.findIndex(r => r.some(c => c === BRICK));
  const brickBottomX = state.x + brick.length - 1;
  return gameHeight > -1 && brickBottomX + 1 > gameHeight;
}

const areAnyBricksColliding = (state: State, brick: Brick): boolean =>
  validBrick(brick).some((r, i) => r.some((c, j) =>
    c === EMPTY
    ? false
    : ((x, y) => state.game[x][y] === c)(i + state.x, j + state.y)
  ));

const collideBrick = (state: State, brick: Brick, isGoingToCollide: boolean): State => {
  const xOffset = isGoingToCollide ? 1 : 0;
  validBrick(brick).forEach((r, i) => {
    r.forEach((c, j) =>
      state.game[i + state.x - xOffset][j + state.y] =
        updatePosition(state.game[i + state.x - xOffset][j + state.y], c)
    );
  });
  state.game = validGame(state.game);
  state.x = 0;
  state.y = (GAME_SIZE / 2) - 1;
  return state;
}

export const collide = (state: State, brick: Brick): [State, Brick] => {
  if (isGoingToLevelWithExistingBricks(state, brick)) {
    return [state, brick];
  }
  if (areAnyBricksColliding(state, brick)) {
    return [collideBrick(state, brick, true), brick];
  }
  return [collideBrick(state, brick, false), randomBrick()];
}

```

```
ck] => {
  const isGoingToCollide =
    isGoingToLevelWithExistingBricks(state, brick) &&
    areAnyBricksColliding(state, brick);

  const isOnBottom = state.x + validBrick(brick).length > GAME_S
    IZE- 1;

  if (isGoingToCollide || isOnBottom) {
    state = collideBrick(state, brick, isGoingToCollide);
    brick = randomBrick();
  }

  return [state, brick];
}
```

rotation.ts


```
import { GAME_SIZE, BRICK_SIZE, EMPTY } from './constants';
import { State, Brick, Key } from './interfaces';

const rightOffsetAfterRotation = (state: State, brick: Brick, rotatedBrick: Brick) =>
  (state.y + rotatedBrick.length === GAME_SIZE + 1) && brick.every(e => e[2] === EMPTY) ? 1 : 0;

const leftOffsetAfterRotation = (game: State) => game.y < 0 ? 1 : 0;

const emptyBrick = (): Brick => Array(BRICK_SIZE).fill(EMPTY).map(e => Array(BRICK_SIZE).fill(EMPTY));

const rotateBrick = (state: State, brick: Brick, rotatedBrick: Brick): [State, Brick] => (
  brick.forEach((r, i) => r.forEach((c, j) => rotatedBrick[j][brick[0].length - 1 - i] = c)),
  state.y -= rightOffsetAfterRotation(state, brick, rotatedBrick),
  state.y += leftOffsetAfterRotation(state),
  [state, rotatedBrick]
)

export const rotate = (state: State, brick: Brick, key: Key): [State, Brick] =>
  key.code === 'ArrowUp' ? rotateBrick(state, brick, emptyBrick()) : [state, brick]
```

keyboard.ts

```
import { GAME_SIZE } from './constants';
import { State, Brick, Key } from './interfaces';

const xOffset = (brick: Brick, columnIndex: number) => brick.every(
  e => e[columnIndex] === 0) ? 1 : 0;

export const handleKeyPress = (state: State, brick: Brick, key:
  Key): State => (
  state.x += key.code === 'ArrowDown'
    ? 1
    : 0,
  state.y += key.code === 'ArrowLeft' && state.y > 0 - xOffset(b
    rick, 0)
    ? -1
    : key.code === 'ArrowRight' && state.y < GAME_SIZE - 3 + xof
    fset(brick, 2)
    ? 1
    : 0,
  state
);

export const resetKey = key => key.code = undefined;
```

html-renderer.ts

```

import { BRICK } from './constants';
import { State, Brick } from './interfaces';
import { updatePosition, validGame, validBrick, clearGame } from
  './game';

const createElem = (column: number): HTMLElement => (elem =>
  (
    elem.style.display = 'inline-block',
    elem.style.marginLeft = '3px',
    elem.style.height = '6px',
    elem.style.width = '6px',
    elem.style['background-color'] = column === BRICK
      ? 'green'
      : 'aliceblue',
    elem
  ))(document.createElement('div'))

export const render = (state: State, brick: Brick): void => {
  const gameFrame = clearGame();

  state.game.forEach((r, i) => r.forEach((c, j) => gameFrame[i][j]
  ] = c));
  validBrick(brick).forEach((r, i) =>
    r.forEach((c, j) => gameFrame[i + state.x][j + state.y] =
      updatePosition(gameFrame[i + state.x][j + state.y], c)));

  document.body.innerHTML = `score: ${state.score} <br/>`;
  validGame(gameFrame).forEach(r => {
    const rowContainer = document.createElement('div');
    r.forEach(c => rowContainer.appendChild(createElem(c)));
    document.body.appendChild(rowContainer);
  });
}

export const renderGameOver = () => document.body.innerHTML += '
<br/>GAME OVER!';

```

interfaces.ts

```
export interface State {  
  game: number[][];  
  x: number;  
  y: number;  
  score: number;  
}  
  
export interface Key {  
  code: string;  
}  
  
export type Brick = number[][];
```

constants.ts

```
export const GAME_SIZE = 10;  
export const BRICK_SIZE = 3;  
export const EMPTY = 0;  
export const BRICK = 1;
```


Operators Used

- [combineLatest](#)
- [finalize](#)
- [fromEvent](#)
- [interval](#)
- [map](#)
- [of](#)
- [pluck](#)
- [scan](#)
- [startWith](#)
- [takeWhile](#)
- [tap](#)

Type Ahead

By [adamlubek](#)


This recipe demonstrates the creation of type ahead client side code.

 Ultimate Courses™

**Fast-track your RxJS skills to build
any project or get hired at a top company.**

Change your life and career - we will get you there!

Show me how! →



with Brian Troncone
Creator of [LearnRxJS.io](#)

Example Code

([StackBlitz](#))

Get continents

```
// RxJS v6+
import { fromEvent, of } from 'rxjs';
import {
  debounceTime,
  distinctUntilChanged,
  map,
  switchMap,
  tap
} from 'rxjs/operators';

const getContinents = keys =>
[
  'africa',
  'antarctica',
  'asia',
  'australia',
  'europe',
  'north america',
  'south america'
].filter(e => e.indexOf(keys.toLowerCase()) > -1);

const fakeContinentsRequest = keys =>
of(getContinents(keys)).pipe(
  tap(_ => console.log(`API CALL at ${new Date()}`))
);

fromEvent(document.getElementById('type-ahead'), 'keyup')
  .pipe(
    debounceTime(200),
    map((e: any) => e.target.value),
    distinctUntilChanged(),
    switchMap(fakeContinentsRequest),
    tap(c => (document.getElementById('output').innerText = c.join('\n')))
  )
  .subscribe();
```

html

Get continents

```
<input id="type-ahead" />
<hr />
<div id="output"></div>
```

Operators Used

- [debounceTime](#)
- [distinctUntilChanged](#)
- [fromEvent](#)
- [map](#)
- [of](#)
- [switchMap](#)
- [tap](#)

Concepts

Short explanations of common RxJS scenarios and use-cases.

Contents

- [RxJS Primer](#)
- [RxJS v5 -> v6 Upgrade](#)
- [Time Based Operator Comparison](#)
- [Understanding Operator Imports](#)

RxJS Primer

Brand new to RxJS? In this article we will take a crash course through all the major concepts you will need to begin getting a grasp on, and start being productive with RxJS. Hold on tight and let's get started!

What is an Observable?

An observable represents a stream, or source of data that can arrive over time. You can create an observable from nearly anything, but the most common use case in RxJS is from events. This can be anything from mouse moves, button clicks, input into a text field, or even route changes. The easiest way to create an observable is through the built in creation functions. For example, we can use the `fromEvent` helper function to create an observable of mouse click events:

```
// import the fromEvent operator
import { fromEvent } from 'rxjs';

// grab button reference
const button = document.getElementById('myButton');

// create an observable of button clicks
const myObservable = fromEvent(button, 'click');
```

At this point we have an observable but it's not doing anything. **This is because observables are cold, or do not activate a producer (like wiring up an event listener), until there is a...**

Subscription

Subscriptions are what set everything in motion. You can think of this like a faucet, you have a stream of water ready to be tapped (observable), someone just needs to turn the handle. In the case of observables, that role belongs to the `subscriber`.

To create a subscription, you call the `subscribe` method, supplying a function (or object) - also known as an `observer`. This is where you can decide how to **react**(-ive programming) to each event. Let's walk through what happens in the previous scenario when a subscription is created:

```
// import the fromEvent operator
import { fromEvent } from 'rxjs';

// grab button reference
const button = document.getElementById('myButton');

// create an observable of button clicks
const myObservable = fromEvent(button, 'click');

// for now, let's just log the event on each click
const subscription = myObservable.subscribe(event => console.log(event));
```

In the example above, calling `myObservable.subscribe()` will:

1. Set up an event listener on our button for click events.
2. Call the function we passed to the subscribe method (observer) on each click event.
3. Return a subscription object with an `unsubscribe` which contains clean up logic, like removing appropriate event listeners.

The subscribe method also accepts an object map to handle the case of error or completion. You probably won't use this as much as simply supplying a function, but it's good to be aware of should the need arise:

```
// instead of a function, we will pass an object with next, error, and complete methods
const subscription = myObservable.subscribe({
  // on successful emissions
  next: event => console.log(event),
  // on errors
  error: error => console.log(error),
  // called once on completion
  complete: () => console.log('complete!')
});
```

It's important to note that each subscription will create a new execution context. This means calling `subscribe` a second time will create a new event listener:

```
// addEventListener called
const subscription = myObservable.subscribe(event => console.log(event));

// addEventListener called again!
const secondSubscription = myObservable.subscribe(event => console.log(event));

// clean up with unsubscribe
subscription.unsubscribe();
secondSubscription.unsubscribe();
```

By default, a subscription creates a one on one, one-sided conversation between the observable and observer. This is like your boss (the observable) yelling (emitting) at you (the observer) for merging a bad PR. This is also known as **unicasting**. If you prefer a conference talk scenario - one observable, many observers - you will take a different approach which includes **multicasting** with `Subjects` (either explicitly or behind the scenes). More on that in a future article!

It's worth noting that when we discuss an Observable source emitting data to observers, this is a push based model. The source doesn't know or care what subscribers do with the data, it simply pushes it down the line.

While working on a stream of events is nice, it's only so useful on its own. **What makes RxJS the "lodash for events" are its...**

Operators

Operators offer a way to manipulate values from a source, returning an observable of the transformed values. Many of the RxJS operators will look familiar if you are used to JavaScripts `Array` methods. For instance, if you want to transform emitted values from an observable source, you can use `map` :

```
import { of } from 'rxjs';
import { map } from 'rxjs/operators';
/*
 * 'of' allows you to deliver values in a sequence
 * In this case, it will emit 1,2,3,4,5 in order.
 */
const dataSource = of(1, 2, 3, 4, 5);

// subscribe to our source observable
const subscription = dataSource
  .pipe(
    // add 1 to each emitted value
    map(value => value + 1)
  )
// log: 2, 3, 4, 5, 6
.subscribe(value => console.log(value));
```

Or if you want to filter for specific values, you can use `filter` :

```
import { of } from 'rxjs';
import { filter } from 'rxjs/operators';

const dataSource = of(1, 2, 3, 4, 5);

// subscribe to our source observable
const subscription = dataSource
  .pipe(
    // only accept values 2 or greater
    filter(value => value >= 2)
  )
  // log: 2, 3, 4, 5
  .subscribe(value => console.log(value));
```

In practice, if there is a problem you need to solve, it's more than likely **there is an operator for that**. And while the sheer number of operators can be overwhelming as you begin your RxJS journey, you can narrow it down to a small handful (and we will) to start being effective. Over time, you will come to appreciate the flexibility of the operator library when obscure scenarios inevitably arrive.

One thing you may have noticed in the example above, is operators exist within a...

Pipe

The `pipe` function is the assembly line from your observable data source through your operators. Just like raw material in a factory goes through a series of stops before it becomes a finished product, source data can pass through a `pipe` -line of operators where you can manipulate, filter, and transform the data to fit your use case. It's not uncommon to use 5 (or more) operators within an observable chain, contained within the `pipe` function.

For instance, a typeahead solution built with observables may use a group of operators to optimize both the request and display process:

```
// observable of values from a text box, pipe chains operators together
inputValue
  .pipe(
    // wait for a 200ms pause
    debounceTime(200),
    // if the value is the same, ignore
    distinctUntilChanged(),
    // if an updated value comes through while request is still
    // active cancel previous request and 'switch' to new observable
    switchMap(searchTerm => typeaheadApi.search(searchTerm))
  )
// create a subscription
.subscribe(results => {
  // update the dom
});
```

But how do you know which operator fits your use-case? The good news is...

Operators can be grouped into common categories

The first stop when looking for the correct operator is finding a related category. Need to filter data from a source? Check out the `filtering` operators. Trying to track down a bug, or debug the flow of data through your observable stream? There are `utility` operators that will do the trick. **The operator categories include...**

Creation operators

These operators allow the creation of an observable from nearly anything. From generic to specific use-cases you are free to turn everything into a stream.

For example, suppose we are creating a progress bar as a user scrolls through an article. We could turn the scroll event into a stream by utilizing the `fromEvent` operator:

```
fromEvent(scrollContainerElement, 'scroll')
  .pipe(
    // we will discuss cleanup strategies like this in future ar
    ticle
    takeUntil(userLeavesArticle)
  )
  .subscribe(event => {
    // calculate and update DOM
  });
```

The most commonly used creation operators are `of` , `from` , and `fromEvent` .

Combination operators

The combination operators allow the joining of information from multiple observables. Order, time, and structure of emitted values is the primary variation among these operators.

For example, we can combine updates from multiple data sources to perform a calculation:

```
// give me the last emitted value from each source, whenever eit
her source emits
combineLatest(sourceOne, sourceTwo).subscribe(
  ([latestValueFromSourceOne, latestValueFromSourceTwo]) => {
    // perform calculation
  }
);
```

The most commonly used combination operators are `combineLatest` , `concat` , `merge` , `startWith` , and `withLatestFrom` .

Error handling operators

The error handling operators provide effective ways to gracefully handle errors and perform retries, should they occur.

For example, we can use `catchError` to safeguard against failed network requests:

```
source
  .pipe(
    mergeMap(value => {
      return makeRequest(value).pipe(
        catchError(handleErrorByReturningObservable)
      );
    })
  )
  .subscribe(value => {
    // take action
  });
```

The most commonly used error handling operators is `catchError` .

Filtering operators

The filtering operators provide techniques for accepting - or declining - values from an observable source and dealing with backpressure, or the build up of values within a stream.

For example, we can use the `take` operator to capture only the first 5 emitted values from a source:

```
source.pipe(take(5)).subscribe(value => {
  // take action
});
```

The most commonly used filtering operators are `debounceTime` , `distinctUntilChanged` , `filter` , `take` , and `takeUntil` .

Multicasting operators

In RxJS observables are cold, or unicast (one source per subscriber) by default. These operators can make an observable hot, or multicast, allowing side-effects to be shared among multiple subscribers.

For example, we may want late subscribers to share, and receive the last emitted value from an active source:

```
const source = data.pipe(shareReplay());

const firstSubscriber = source.subscribe(value => {
  // perform some action
});

// sometime later...

// second subscriber gets last emitted value on subscription, shares execution context with 'firstSubscriber'
const secondSubscriber = source.subscribe(value => {
  // perform some action
});
```

The most commonly used multicasting operator is `shareReplay`.

Transformation operators

Transforming values as they pass through an operator chain is a common task. These operators provide transformation techniques for nearly any use-case you will encounter.

For example, we may want to accumulate a state object from a source over time, similar to [Redux](#):

```
source
  .pipe(
    scan((accumulatedState, currentState) => {
      return { ...accumulatedState, ...currentState };
    })
  )
  .subscribe();
```

The most commonly used transformation operators are `concatMap` , `map` , `mergeMap` , `scan` , and `switchMap` .

Operators have common behaviors

While operators can be grouped into common categories, operators within a category often share common behavior. By recognizing this common behavior you can start creating a *'choose your own operator' tree* in your mind.

For instance, a large amount of operators can be grouped into...

Operators that flatten

Or, in other words, operators that manage the subscription of an inner observable, emitting those values into a single observable source. One common use case for flattening operators is handling HTTP requests in an observable or promise based API, but that is really just scratching the surface:

```
fromEvent(button, 'click')
  .pipe(
    mergeMap(value => {
      // this 'inner' subscription is managed by mergeMap, with
      // response value emitted to observer
      return makeHttpRequest(value);
    })
  )
  .subscribe(response => {
    // do something
  });
```

We can then divide the flattening operators into common behaviors like...

Operators that `switch`

Like a light switch, `switch` based operators will turn off (unsubscribe) the current observable and turn on a new observable on emissions from the source. Switch operators are useful in situations you don't want (or need) more than one active observable at a time:

```
inputValueChanges
  // only the last value is important, if new value comes through
  // cancel previous request / observable
  .pipe(
    // make GET request for data
    switchMap(requestObservable)
  )
  .subscribe();
```

Switch based operators include `switchAll` , `switchMap` , and `switchMapTo` .

Operators that `concat`

Comparable to a line at the ATM machine, the next transaction can't begin until the previous completes. In observable terms, only one subscription will occur at a time, in order, triggered by the completion of the previous. These are useful in situations where order of execution is important:

```
concat(
  firstObservable,
  // will begin when 'firstObservable` completes
  secondObservable,
  // will begin when 'secondObservable` completes
  thirdObservable
).subscribe(values => {
  // take action
});
```

Concat based operators include `concat` , `concatAll` , `concatMap` , and `concatMapTo` .

Operators that `merge`

Like your merging lane on the interstate, merge based operators support multiple active observables flowing into one lane in a first come first serve basis. Merge operators are useful in situations where you want to trigger an action when an event from one of many sources occurs:

```
merge(firstObservable, secondObservable)
  // any emissions from first or second observable as they occur
  .pipe(mergeMap(saveActivity))
  .subscribe();
```

Merge based operators include `merge` , `mergeMap` , `mergeMapTo` and `mergeAll` .

Other similarities between operators

There are also operators that share a similar goal but offer flexibility in their triggers. For instance, for unsubscribing from an observable after a specific condition is met, we could use:

1. `take` when we know we only ever want `n` values.
2. `takeLast` when you just want the last `n` values.
3. `takeWhile` when we have a predicate expression to supply.
4. `takeUntil` when we only want the source to remain active until another source emits.

While the number of RxJS operators can seem overwhelming at first, these common behaviors and patterns can bridge the gap rather quickly while learning RxJS.

What does this get me?

As you become more familiar with push based programming through Observables, you can begin to model all async behavior in your applications through observable streams. This opens up simple solutions and flexibility for notably complex behavior.

For instance, suppose we wanted to make a request which saved user activity when they answered a quiz question. Our initial implementation may use the `mergeMap` operator, which fires off a save request on each event:

```
const formEvents = fromEvent(formField, 'click');
const subscription = formEvents
  .pipe(
    map(convertToAppropriateValue),
    mergeMap(saveRequest)
  )
  .subscribe();
```

Later, it's determined that we need to ensure order of these saves. Armed with the knowledge of operator behavior from above, instead of implementing a complex queueing system we can instead replace the `mergeMap` operator with `concatMap` and push up our changes:

```
const formEvents = fromEvent(formField, 'click');
const subscription = formEvents
  .pipe(
    map(convertToAppropriateValue),
    // now the next request won't begin until the previous completes
    concatMap(saveRequest)
  )
  .subscribe();
```

With the change of one word we are now queueing our event requests, and this is just scratching the surface of what is possible!

Keep Going!

Learning RxJS can be intimidating, but it's a path I promise is worth the investment. If some of these concepts are still fuzzy (or make no sense at all!), don't worry! It will all click soon.

Start checking out the operators on the left hand side of the site for common examples and use-cases, as well as the additional [introductory resources](#) we have collected from across the web. Good luck and enjoy your journey to becoming a reactive programming expert!

RxJS v5 -> v6 Upgrade

Preparing to upgrade from RxJS v5 to v6? Here are some resources you might find handy:

RxJS-TsLint

TsLint rules for migration to RxJS 6. Auto-update project for new import paths and transition to pipeable operators.

RxJS v5.x to v6 Update Guide

Comprehensive guide for updating your project from RxJS v5 to 6

Interactive Comparison of RxJS v5 and v6 Code

Demonstrates differences between v5 and v6 code, as well as RxJS examples utilizing the experimental [pipeline operator](#).

Pipeable Operators

Explanation and examples of utilizing pipeable operators.

Time based operators comparison

RxJS offers a rich selection of time based operators but this diversity can come at cost when choosing the right operator for a task at hand. Below is a visual comparison of popular time based operators.

Compared operators:

- [auditTime](#)
- [bufferTime](#)
- [debounceTime](#)
- [sampleTime](#)
- [throttleTime](#)

([Stackblitz](#))

```
/*
interval      ^0-----1-----2-----3-----4-----5-----6
-----7-----8-----9-----|
auditTime     ^-----3-----
-----7-----|
bufferTime    ^-----[0,1]-----[2,3,4]--
-----[5,6,7]-[8,9]--|
debounceTime  ^-----
-----9-----|
sampleTime    ^-----1-----4-----
-----7-----|
throttleTime  ^0-----4-----
-----8-----|
*/

// RxJS v6+
import { interval, merge } from 'rxjs';
import { auditTime, bufferTime, debounceTime, sampleTime, throttleTime, tap, take } from 'rxjs/operators';

const takeValue = 10;
const intrvl = 1000;
const time = 3000;
```



```
const intervalled = (operator, operatorName) =>
  interval(intrvl)
    .pipe(
      take(takeValue),
      operator,
      tap(x => console.log(`${operatorName}:${x}`))
    );

merge(
  interval(intrvl).pipe(take(takeValue), tap(v => console.log(`i
: ${v}`))),
  intervalled(auditTime(time), "auditTime"),
  intervalled(bufferTime(time), "bufferTime"),
  intervalled(debounceTime(time), "debounceTime"),
  intervalled(sampleTime(time), "sampleTime"),
  intervalled(throttleTime(time), "throttleTime")
).subscribe();

// output
/*
0
throttleTime:0
1
2
bufferTime:0,1
sampleTime:1
3
auditTime:3
4
throttleTime:4
5
bufferTime:2,3,4
sampleTime:4
6
7
auditTime:7
8
bufferTime:5,6,7
sampleTime:7
throttleTime:8
9
```

```
bufferTime:8,9  
debounceTime:9
```

```
* /
```

Understanding Operator Imports

A problem you may have run into in the past when consuming or creating a public library that depends on RxJS is handling operator inclusion. The most predominant way to include operators in your project is to import them like below:

```
import 'rxjs/add/operator/take';
```

This adds the imported operator to the `Observable` prototype for use throughout your project:

[\(Source\)](#)

```
import { Observable } from '../Observable';
import { take } from '../operator/take';

Observable.prototype.take = take;

declare module '../Observable' {
  interface Observable<T> {
    take: typeof take;
  }
}
```

This method is generally *OK* for private projects and modules, the issue arises when you are using these imports in say, an [npm](#) package or library to be consumed throughout your organization.

A Quick Example

To see where a problem can spring up, let's imagine **Person A** is creating a public Angular component library. In this library you need a few operators so you add the typical imports:

some-public-library.ts

```
import 'rxjs/add/operator/take';  
import 'rxjs/add/operator/concatMap';  
import 'rxjs/add/operator/switchMap';
```

Person B comes along and includes your library. They now have access to these operators even though they did not personally import them. *Probably not a huge deal but it can be confusing.* You use the library and operators, life goes on...

A month later **Person A** decides to update their library. They no longer need

`switchMap` or `concatMap` so they remove the imports:

some-public-library.ts

```
import 'rxjs/add/operator/take';
```

Person B upgrades the dependency, builds their project, which now fails. They never included `switchMap` or `concatMap` themselves, it was **just working** based on the inclusion of a 3rd party dependency. If you were not aware this could be an issue it may take a bit to track down.

The Solution

Instead of importing operators like:

```
import 'rxjs/add/operator/take';
```

We can instead import them like:

```
import { take } from 'rxjs/operator/take';
```

This keeps them off the `Observable` prototype and let's us call them directly:

```
import { take } from 'rxjs/operator/take';
import { of } from 'rxjs/observable/of';

take.call(
  of(1, 2, 3),
  2
);
```

This quickly gets **ugly** however, imagine we have a longer chain:

```
import { take } from 'rxjs/operator/take';
import { map } from 'rxjs/operator/map';
import { of } from 'rxjs/observable/of';

map.call(
  take.call(
    of(1, 2, 3),
    2
  ),
  val => val + 2
);
```

Pretty soon we have a block of code that is near impossible to understand. How can we get the best of both worlds?

Pipeable Operators

RxJS now comes with a `pipe` helper on `Observable` that alleviates the pain of not having operators on the prototype. We can take the ugly block of code from above:

```
import { take, map } from 'rxjs/operators';
import { of } from 'rxjs/observable/of';

map.call(
  take.call(
    of(1, 2, 3),
    2
  ),
  val => val + 2
);
```

And transform it into:

```
import { take, map } from 'rxjs/operators';
import { of } from 'rxjs/observable/of';

of(1, 2, 3)
  .pipe(
    take(2),
    map(val => val + 2)
  );
```

Much easier to read, right? This also has the benefit of greatly reducing the RxJS bundle size in your application. For more on this, check out [Ashwin Sureshkumar's](#) excellent article [Reduce Angular app bundle size using lettable operators](#).