**buffer**

//Create an observable that emits a value every second

const myInterval = Rx.Observable.interval(1000);

//Create an observable that emits every time document is clicked

const bufferBy = Rx.Observable.fromEvent(document, 'click');

/\*

Collect all values emitted by our interval observable until we click document. This will cause the bufferBy Observable to emit a value, satisfying the buffer. Pass us all collected values since last buffer as an array.

\*/

const myBufferedInterval = myInterval.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));

**bufferCount**

#####signature: bufferCount<T>(bufferSize: number, startBufferEvery: number = null): Observable<T[]> *The gist: Collect output values until specified number is fulfilled then hand them over. Repeat...*

([demo](http://jsbin.com/xibixetiqa/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-bufferCount))

//Create an observable that emits a value every second

const myInterval = Rx.Observable.interval(1000);

//After three values are emitted, pass on as an array of buffered values

const bufferThree = myInterval.bufferCount(3);

//Print values to console

//ex. output [0,1,2]...[3,4,5]

const subscribe = bufferThree.subscribe(val => console.log('Buffered Values:', val));

/\*

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 = myInterval.bufferCount(3,1);

//Print values to console

const secondSubscribe = bufferEveryOne.subscribe(val => console.log('Start Buffer Every 1:', val))

**bufferTime**

#####signature: bufferTime(bufferTimeSpan: number, bufferCreationInterval: number, scheduler: Scheduler): Observable<T[]> *The gist: Collect output values until specified time has passed then hand them over. Repeat...*

([demo](http://jsbin.com/gixarikeme/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-bufferTime))

//Create an observable that emits a value every 500ms

const myInterval = Rx.Observable.interval(500);

//After 2 seconds have passed, emit buffered values as an array

const bufferTime = myInterval.bufferTime(2000);

//Print values to console

//ex. output [0,1,2]...[3,4,5,6]

const subscribe = bufferTime.subscribe(val => console.log('Buffered with Time:', val));

/\*

bufferTime also takes second argument, when to start the next buffer (time in ms)

for instance, if we have a bufferTime of 2 seconds but second argument (bufferCreationInterval) of 1 second:

ex. output: [0,1,2]...[1,2,3,4,5]...[3,4,5,6,7]

\*/

const bufferTimeTwo = myInterval.bufferTime(2000,1000);

//Print values to console

const secondSubscribe = bufferTimeTwo.subscribe(val => console.log('Start Buffer Every 1s:', val));

**bufferToggle**

#####signature: bufferToggle(openings: Observable<O>, closingSelector: Function): Observable<T[]> *The gist: Toggle buffer on to catch emitted values from source, toggle buffer off to emit buffered values...*

([demo](http://jsbin.com/relavezugo/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-bufferToggle))

//emit value every second

const sourceInterval = Rx.Observable.interval(1000);

//start first buffer after 5s, and every 5s after

const startInterval = Rx.Observable.interval(5000);

//emit value after 3s, closing corresponding buffer

const closingInterval = val => {

console.log(`Value ${val} emitted, starting buffer! Closing in 3s!`)

return Rx.Observable.interval(3000);

}

//every 5s a new buffer will start, collecting emitted values for 3s then emitting buffered values

const bufferToggleInterval = sourceInterval.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));

**bufferWhen**

#####signature: bufferWhen(closingSelector: function): Observable<T[]> *The gist: Buffer all values until closing selector emits, emit buffered values, repeat...*

([demo](http://jsbin.com/vugerupube/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-bufferWhen))

//emit value every 1 second

const oneSecondInterval = Rx.Observable.interval(1000);

//return an observable that emits value every 5 seconds

const fiveSecondInterval = () => Rx.Observable.interval(5000);

//every five seconds, emit buffered values

const bufferWhenExample = oneSecondInterval.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));

**combineAll**

#####signature: combineAll(project: function): Observable *The gist: Output latest values from inner observables when outer observable completes...*

([demo](http://jsbin.com/boxadasevu/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-combineAll))

//emit after five seconds then complete

const fiveSecondTimer = Rx.Observable.timer(5000);

//once timer (outer observable) fires and completes, latest emitted values from inner observables will be output, in this case there is a single value

const example = fiveSecondTimer.mapTo(Rx.Observable.of('Hello', 'World'));

const combined = example.combineAll();

//ex output: ["Hello"]...["World"]

const subscribe = combined.subscribe(val => console.log('Values from inner observable:', val));

//combineAll also takes a projection function that receives emitted values

const fiveSecondTimer = Rx.Observable.timer(5000);

const example = fiveSecondTimer.mapTo(Rx.Observable.of('Hello', 'Goodbye'));

const combined = example.combineAll(val => `${val} Friend!`);

//ex output: "Hello Friend!"..."Goodbye Friend!"

const subscribeProjected = combined.subscribe(val => console.log('Values Using Projection:', val));

**combineLatest**

#####signature: combineLatest(observables: ...Observable, project: function): Observable *The gist: Given a group of observables, when one emits also emit latest values from each...*

([demo](http://jsbin.com/lumaqanoha/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-combineLatest))

//timerOne emits first value at 1s, then once every 4s

const timerOne = Rx.Observable.timer(1000, 4000);

//timerTwo emits first value at 2s, then once every 4s

const timerTwo = Rx.Observable.timer(2000, 4000)

//timerThree emits first value at 3s, then once every 4s

const timerThree = Rx.Observable.timer(3000, 4000)

//when one timer emits, emit the latest values from each timer as an array

const combined = Rx.Observable

.combineLatest(

timerOne,

timerTwo,

timerThree

);

const subscribe = combined.subscribe(latestValues => {

//grab latest emitted values for timers one, two, and three

const [timerValOne, timerValTwo, timerValThree] = latestValues;

/\*

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}`

);

});

//combineLatest also takes an optional projection function

const combinedProject = Rx.Observable

.combineLatest(

timerOne,

timerTwo,

timerThree,

(one, two, three) => {

return `Timer One (Proj) Latest: ${one},

Timer Two (Proj) Latest: ${two},

Timer Three (Proj) Latest: ${three}`

}

);

//log values

const subscribe = combinedProject.subscribe(latestValuesProject => console.log(latestValuesProject));

**concat**

#####signature: concat(observables: ...\*): Observable *The gist: Like the line at an ATM, the next transaction (subscription) won't start until the previous completes...*

([demo](http://jsbin.com/kenusofudu/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-concat))

//emits 1,2,3

const sourceOne = Rx.Observable.of(1,2,3);

//emits 4,5,6

const sourceTwo = Rx.Observable.of(4,5,6);

//emit values from sourceOne, when complete, subscribe to sourceTwo

const concatSource = sourceOne.concat(sourceTwo);

//output: 1,2,3,4,5,6

const subscribe = concatSource.subscribe(val => console.log('Example 1: Basic concat:', val));

//delay 3 seconds then emit

const delayedSourceOne = sourceOne.delay(3000);

//sourceTwo waits on sourceOne to complete before subscribing

const concatDelayedSource = delayedSourceOne.concat(sourceTwo);

//output: 1,2,3,4,5,6

const subscribeDelayed = concatDelayedSource.subscribe(val => console.log('Example 2: Delayed source one:', val));

//when sourceOne never completes, the subsequent observables never run

const sourceOneNeverComplete = Rx.Observable

.concat(

Rx.Observable.interval(1000),

Rx.Observable.of('This','Never','Runs')

)

//for logging clarity

.delay(5000)

//outputs: 1,2,3,4....

const subscribeNeverComplete = sourceOneNeverComplete.subscribe(val => console.log('Example 3: Source one never completes, second observable never runs:', val));

**concatAll**

#####signature: concatAll(): Observable *The gist: Concat for nested observables (observable of observables), subscribe to each when previous completes and merge emitted values...*

([demo](http://jsbin.com/hayasoxoci/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-concatAll))

//emit a value every 2 seconds

const sourceOne = Rx.Observable.interval(2000);

const example = sourceOne

//for demonstration, add 10 to and return as observable

.map(val => Rx.Observable.of(val + 10))

//merge values from inner observable

.concatAll();

//output: 'Example with Basic Observable 0', 'Example with Basic Observable 2'...

const subscribe = example.subscribe(val => console.log('Example with Basic Observable:', val));

//create and resolve basic promise

const samplePromise = val => new Promise(resolve => resolve(val));

const exampleTwo = sourceOne

.map(val => samplePromise(val))

//merge values from resolved promise

.concatAll();

//output: 'Example with Promise 0', 'Example with Promise 1'...

const subscribeTwo = exampleTwo.subscribe(val => console.log('Example with Promise:', val));

**concatMap**

#####signature: concatMap(project: function, resultSelector: function): Observable *The gist: Map values from source to inner observable, subscribe and emit inner observable values in order...*

*You could also: map -> concatAll*

([demo](http://jsbin.com/dekadarube/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-concatMap))

//emit 'Hello' and 'Goodbye'

const source = Rx.Observable.of('Hello', 'Goodbye');

// map value from source into inner observable, when complete emit result and move to next

const exampleOne = source.concatMap(val => Rx.Observable.of(`${val} World!`));

//output: 'Example One: 'Hello World', Example One: 'Goodbye World'

const subscribe = exampleOne

.subscribe(val => console.log('Example One:', val));

//example with promise

const examplePromise = val => new Promise(resolve => resolve(`${val} World!`));

// map value from source into inner observable, when complete emit result and move to next

const exampleTwo = source.concatMap(val => examplePromise(val))

//output: 'Example w/ Promise: 'Hello World', Example w/ Promise: 'Goodbye World'

const subscribeTwo = exampleTwo

//delay for logging clarity

.delay(1000)

.subscribe(val => console.log('Example w/ Promise:', val));

//result of first param passed to second param selector function before being returned

const exampleWithSelector = source.concatMap(val => examplePromise(val), result => `${result} w/ selector!`);

//output: 'Example w/ Selector: 'Hello w/ Selector', Example w/ Selector: 'Goodbye w/ Selector'

const subscribeThree = exampleWithSelector

//delay for logging clarity

.delay(2000)

.subscribe(val => console.log('Example w/ Selector:', val));

**concatMapTo**

#####signature: concatMapTo(observable: Observable, resultSelector: function): Observable *The gist: When source emits, always subscribe to the same observable, merging together results when complete...*

([demo](http://jsbin.com/caqiruqula/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-concatMapTo))

//emit value every 2 seconds

const interval = Rx.Observable.interval(2000);

const message = Rx.Observable.of('Second(s) elapsed!');

//when interval emits, subscribe to message until complete, merge for result

const example = interval.concatMapTo(message, (time, msg) => `${time} ${msg}`);

//log values

//output: '0 Second(s) elapsed', '1 Second(s) elapsed'

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

//emit value every second for 5 seconds

const basicTimer = Rx.Observable.interval(1000).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 exampleTwo = interval

.concatMapTo(basicTimer,

(firstInterval, secondInterval) => `${firstInterval} ${secondInterval}`

);

/\*

output: 0 0

0 1

0 2

0 3

0 4

1 0

1 1

continued...

\*/

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**count**

#####signature: count(predicate: function): Observable *The gist: Count values emitted from source until complete, optionally base count on predicate...*

([demo](http://jsbin.com/hitemoxica/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-count))

//emit 1,2,3 then complete

const threeItems = Rx.Observable.of(1,2,3);

//when threeItems completes, return count of items emitted

const exampleOne = threeItems.count();

//output: 'Count from Example One: 3'

const subscribe = exampleOne.subscribe(val => console.log(`Count from Example One: ${val}`));

//count of basic array

const myArray = [1,2,3,4,5];

const myObsArray = Rx.Observable.from(myArray);

const exampleTwo = myObsArray.count();

//output: 'Count of Basic Array: 5'

const subscribeTwo = exampleTwo.subscribe(val => console.log(`Count of Basic Array: ${val}`));

//count emitted items from source that satisfy predicate expression

const evensCount = myObsArray.count(val => val % 2 === 0);

//output: 'Count of Even Numbers: 2'

const subscribeThree = evensCount.subscribe(val => console.log(`Count of Even Numbers: ${val}`));

**debounce**

#####signature: debounce(durationSelector: function): Observable *The gist: Throw away all emitted values that take less then the specified time (based on selector function) between output...*

([demo](http://jsbin.com/cofofizopo/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-debounce))

//emit four strings

const example = Rx.Observable.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.debounce(() => Rx.Observable.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));

//emit value every 1 second, ex. 0...1...2

const interval = Rx.Observable.interval(1000);

//raise the debounce time by 200ms each second

const debouncedInterval = interval.debounce(val => Rx.Observable.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 subscribeTwo = debouncedInterval.subscribe(val => console.log(`Example Two: ${val}`));

**debounceTime**

#####signature: debounceTime(dueTime: number, scheduler: Scheduler): Observable *The gist: Throw away all emitted values that take less then the specified time between output...*

([demo](http://jsbin.com/kacijarogi/1/edit?js,console,output) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-debounceTime))

const input = document.getElementById('example');

//for every keyup, map to current input value

const example = Rx.Observable

.fromEvent(input, 'keyup')

.map(i => i.currentTarget.value);

//wait .5s between keyups to emit current value

//throw away all other values

const debouncedInput = example.debounceTime(500);

//log values

const subscribe = debouncedInput.subscribe(val => {

console.log(`Debounced Input: ${val}`);

});

**defaultIfEmpty**

#####signature: defaultIfEmpty(defaultValue: any): Observable *The gist: When observable is empty use given default, or null...*

([demo](http://jsbin.com/ricotitasu/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-defaultIfEmpty))

const empty = Rx.Observable.of();

//emit 'Observable.of() Empty!' when empty, else any values from source

const exampleOne = empty.defaultIfEmpty('Observable.of() Empty!');

//output: 'Observable.of() Empty!'

const subscribe = exampleOne.subscribe(val => console.log(val));

//empty observable

const emptyTwo = Rx.Observable.empty();

//emit 'Observable.empty()!' when empty, else any values from source

const exampleTwo = emptyTwo.defaultIfEmpty('Observable.empty()!');

//output: 'Observable.empty()!'

const subscribe = exampleTwo.subscribe(val => console.log(val));

**delay**

#####signature:  delay(delay: number | Date, scheduler: Scheduler): Observable *The gist: Delay output by specified time...*

([demo](http://jsbin.com/zebatixije/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-delay))

//emit one item

const example = Rx.Observable.of(null);

//delay output of each by an extra second

const message = Rx.Observable.merge(

example.mapTo('Hello'),

example.mapTo('World!').delay(1000),

example.mapTo('Goodbye').delay(2000),

example.mapTo('World!').delay(3000)

);

//output: 'Hello'...'World!'...'Goodbye'...'World!'

const subscribe = message.subscribe(val => console.log(val));

**delayWhen**

#####signature: delayWhen(selector: Function, sequence: Observable): Observable *The gist: Delay output by specified time, determined by provided function...*

([demo](http://jsbin.com/topohekuje/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-delayWhen))

//emit value every second

const message = Rx.Observable.interval(1000);

//emit value after five seconds

const delayForFiveSeconds = () => Rx.Observable.timer(5000);

//after 5 seconds, start emitting delayed interval values

const delayWhenExample = message.delayWhen(delayForFiveSeconds);

//log values, delayed for 5 seconds

//ex. output: 5s....1...2...3

const subscribe = delayWhenExample.subscribe(val => console.log(val));

**dematerialize**

#####signature: dematerialize(): Observable *The gist: Turn notification objects into notification values...*

([demo](http://jsbin.com/vafedocibi/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-dematerialize))

//emit next and error notifications

const source = Rx.Observable

.from([

Rx.Notification.createNext('SUCCESS!'),

Rx.Notification.createError('ERROR!')

])

//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}`)

});

**distinctUntilChanged**

#####signature:  distinctUntilChanged(compare: function): Observable *The gist: Only emit when the next value is different then the last...*

([demo](http://jsbin.com/wuhumodoha/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-distinctUntilChanged))

//only output distinct values, based on the last emitted value

const myArrayWithDuplicatesInARow = Rx.Observable

.from([1,1,2,2,3,1,2,3]);

const distinctSub = myArrayWithDuplicatesInARow

.distinctUntilChanged()

//output: 1,2,3,1,2,3

.subscribe(val => console.log('DISTINCT SUB:', val));

const nonDistinctSub = myArrayWithDuplicatesInARow

//output: 1,1,2,2,3,1,2,3

.subscribe(val => console.log('NON DISTINCT SUB:', val));

const sampleObject = {name: 'Test'};

const myArrayWithDuplicateObjects = Rx.Observable.from([sampleObject, sampleObject, sampleObject]);

//only out distinct objects, based on last emitted value

const nonDistinctObjects = myArrayWithDuplicateObjects

.distinctUntilChanged()

//output: 'DISTINCT OBJECTS: {name: 'Test'}

.subscribe(val => console.log('DISTINCT OBJECTS:', val));

**do**

#####signature: do(nextOrObserver: function, error: function, complete: function): Observable *The gist: Transparently perform actions, such as logging...*

([demo](http://jsbin.com/jimazuriva/1/edit?js,console) | [official docs](https://github.com/ReactiveX/rxjs/blob/master/src/operator/do.ts))

const source = Rx.Observable.of(1,2,3,4,5);

//transparently log values from source with 'do'

const example = source

.do(val => console.log(`BEFORE MAP: ${val}`))

.map(val => val + 10)

.do(val => console.log(`AFTER MAP: ${val}`));

//'do' does not transform values

//output: 11...12...13...14...15

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

**every**

#####signature: every(predicate: function, thisArg: any): Observable *The gist: Does every emitted item pass a condition?...*

([demo](http://jsbin.com/mafacebuwu/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-every))

//emit 5 values

const source = Rx.Observable.of(1,2,3,4,5);

const example = source

//is every value even?

.every(val => val % 2 === 0)

//output: false

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

//emit 5 values

const allEvens = Rx.Observable.of(2,4,6,8,10);

const exampleTwo = allEvens

//is every value even?

.every(val => val % 2 === 0);

//output: true

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**expand**

#####signature: expand(project: function, concurrent: number, scheduler: Scheduler): Observable *The gist: Recursively call provided function...*

([demo](http://jsbin.com/fuxocepazi/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-expand))

//emit 2

const source = Rx.Observable.of(2);

const example = source

//recursively call supplied function

.expand(val => {

//2,3,4,5,6

console.log(`Passed value: ${val}`);

//3,4,5,6

return Rx.Observable.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}`));

**filter**

#####signature: filter(select: Function, thisArg: any): Observable *The gist: Only return values that pass the provided condition...*

([demo](http://jsbin.com/gaqojobove/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-filter))

//emit (1,2,3,4,5)

const source = Rx.Observable.from([1,2,3,4,5]);

//filter out non-even numbers

const example = source.filter(num => num % 2 === 0);

//output: "Even number: 2", "Even number: 4"

const subscribe = example.subscribe(val => console.log(`Even number: ${val}`));

//emit ({name: 'Joe', age: 31}, {name: 'Bob', age:25})

const sourceTwo = Rx.Observable.from([{name: 'Joe', age: 31}, {name: 'Bob', age:25}]);

//filter out people with age under 30

const exampleTwo = sourceTwo.filter(person => person.age >= 30);

//output: "Over 30: Joe"

const subscribeTwo = exampleTwo.subscribe(val => console.log(`Over 30: ${val.name}`));

//emit every second

const sourceThree = Rx.Observable.interval(1000);

//filter out all values until interval is greater than 5

const exampleThree = sourceThree.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 subscribeThree = exampleThree.subscribe(val => console.log(`Number greater than 5: ${val}`));

**first**

#####signature: first(predicate: function, select: function) *The gist: Emit the first value, or the first to pass condition...*

([demo](http://jsbin.com/poloquxuja/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-first))

const source = Rx.Observable.from([1,2,3,4,5]);

//no arguments, emit first value

const example = source.first();

//output: "First value: 1"

const subscribe = example.subscribe(val => console.log(`First value: ${val}`));

//emit first item to pass test

const exampleTwo = source.first(num => num === 5);

//output: "First to pass test: 5"

const subscribeTwo = exampleTwo.subscribe(val => console.log(`First to pass test: ${val}`));

//using optional projection function

const exampleThree = source.first(num => num % 2 === 0,

(result, index) => `First even: ${result} is at index: ${index}`);

//output: "First even: 2 at index: 1"

const subscribeThree = exampleThree.subscribe(val => console.log(val));

**groupBy**

#####signature: groupBy(keySelector: Function, elementSelector: Function): Observable *The gist: Group into observables by given value...*

([demo](http://jsbin.com/zibomoluru/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-groupBy))

const people = [{name: 'Sue', age:25},{name: 'Joe', age: 30},{name: 'Frank', age: 25}, {name: 'Sarah', age: 35}];

//emit each person

const source = Rx.Observable.from(people);

//group by age

const example = source

.groupBy(person => person.age)

//return as array of each group

.flatMap(group => group.reduce((acc, curr) => [...acc, ...curr], []))

/\*

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));

**ignoreElements**

#####signature: ignoreElements(): Observable *The gist: Ignore everything but complete and error...*

([demo](http://jsbin.com/luyufeviqu/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-ignoreElements))

//emit value every 100ms

const source = Rx.Observable.interval(100);

//ignore everything but complete

const example = source

.take(5)

.ignoreElements();

//output: "COMPLETE!"

const subscribe = example.subscribe(

val => console.log(`NEXT: ${val}`),

val => console.log(`ERROR: ${val}`),

() => console.log('COMPLETE!')

);

//ignore everything but error

const error = source

.flatMap(val => {

if(val === 4){

return Rx.Observable.throw(`ERROR AT ${val}`);

}

return Rx.Observable.of(val);

})

.ignoreElements();

//output: "ERROR: ERROR AT 4"

const subscribeTwo = error.subscribe(

val => console.log(`NEXT: ${val}`),

val => console.log(`ERROR: ${val}`),

() => console.log('SECOND COMPLETE!')

);

**last**

#####signature: last(predicate: function): Observable *The gist: Emit last item or last to pass test...*

([demo](http://jsbin.com/xidufijuku/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-last))

const source = Rx.Observable.from([1,2,3,4,5]);

//no arguments, emit last value

const example = source.last();

//output: "Last value: 5"

const subscribe = example.subscribe(val => console.log(`Last value: ${val}`));

//emit last even number

const exampleTwo = source.last(num => num % 2 === 0);

//output: "Last to pass test: 4"

const subscribeTwo = exampleTwo.subscribe(val => console.log(`Last to pass test: ${val}`));

**let**

#####signature: let(function): Observable *The gist: let me have the whole observable...*

([demo](http://jsbin.com/bivisofuxe/edit?js,console) | [official docs](https://github.com/Reactive-Extensions/RxJS/blob/master/doc/api/core/operators/let.md))

const myArray = [1,2,3,4,5];

const myObservableArray = Rx.Observable.fromArray(myArray);

//demonstrating the difference between let and other operators

const test = myObservableArray

.map(val => val + 1)

//this fails, 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 letTest = myObservableArray

.map(val => val + 1)

//'let' me have the entire observable

.let(obs => obs.map(val => val + 2))

.subscribe(val => console.log('VALUE FROM ARRAY WITH let: ', val));

//let provides flexibility to add multiple operators to source observable then return

const letTestThree = myObservableArray

.map(val => val + 1)

.let(obs => obs

.map(val => val + 2)

//also, just return evens

.filter(val => val % 2 === 0)

)

.subscribe(val => console.log('let WITH MULTIPLE OPERATORS: ', val));

//pass in your own function to add operators to observable

const obsArrayPlusYourOperators = (yourAppliedOperators) => {

return myObservableArray

.map(val => val + 1)

.let(yourAppliedOperators)

};

const addTenThenTwenty = obs => obs.map(val => val + 10).map(val => val + 20);

const letTestFour = obsArrayPlusYourOperators(addTenThenTwenty)

.subscribe(val => console.log('let FROM FUNCTION:', val));

**map**

#####signature: map(project: Function, thisArg: any): Observable *The gist: Apply projection to each element...*

([demo](http://jsbin.com/vegagizedo/1/edit?js,console) | [official docs](http://reactivex-rxjs5.surge.sh/function/index.html#static-function-map))

//emit (1,2,3,4,5)

const source = Rx.Observable.from([1,2,3,4,5]);

//add 10 to each value

const example = source.map(val => val + 10);

//output: 11,12,13,14,15

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

//emit ({name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name: 'Ryan', age: 50})

const sourceTwo = Rx.Observable.from([{name: 'Joe', age: 30}, {name: 'Frank', age: 20},{name: 'Ryan', age: 50}]);

//grab each persons name

const exampleTwo = sourceTwo.map(person => person.name);

//output: "Joe","Frank","Ryan"

const subscribe = exampleTwo.subscribe(val => console.log(val));

**mapTo**

#####signature: mapTo(value: any): Observable *The gist: Map to a constant value every time...*

([demo](http://jsbin.com/yazusehahu/1/edit?js,console,output) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-mapTo))

//emit value every two seconds

const source = Rx.Observable.interval(2000);

//map all emissions to one value

const example = source.mapTo('HELLO WORLD!');

//output: 'HELLO WORLD!'...'HELLO WORLD!'...'HELLO WORLD!'...

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

//emit every click on document

const clickSource = Rx.Observable.fromEvent(document, 'click');

//map all emissions to one value

const exampleTwo = clickSource.mapTo('GOODBYE WORLD!');

//output: (click)'GOODBYE WORLD!'...

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**merge**

#####signature: merge(input: Observable): Observable *The gist: Squish outputs from multiple observables into a single source...*

([demo](http://jsbin.com/wicubemece/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-merge))

//emit every 2.5 seconds

const first = Rx.Observable.interval(2500);

//emit every 2 seconds

const second = Rx.Observable.interval(2000);

//emit every 1.5 seconds

const third = Rx.Observable.interval(1500);

//emit every 1 second

const fourth = Rx.Observable.interval(1000);

//emit outputs from one observable

const example = Rx.Observable.merge(

first.mapTo('FIRST!'),

second.mapTo('SECOND!'),

third.mapTo('THIRD'),

fourth.mapTo('FOURTH')

);

//output: "FOURTH", "THIRD", "SECOND!", "FOURTH", "FIRST!", "THIRD", "FOURTH"

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

//used as instance method

const exampleTwo = first.merge(fourth);

//output: 0,1,0,2....

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**mergeMap**

#####signature: mergeMap(project: function: Observable, resultSelector: function: any, concurrent: number): Observable *The gist: Map values from source to inner observable, flatten output...*

*You could also: map -> mergeAll*

([demo](http://jsbin.com/haxobidino/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-mergeMap))

//emit 'Hello'

const source = Rx.Observable.of('Hello');

//map to inner observable and flatten

const example = source.mergeMap(val => Rx.Observable.of(`${val} World!`));

//output: 'Hello World!'

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

//mergeMap also emits result of promise

const myPromise = val => new Promise(resolve => resolve(`${val} World From Promise!`));

//map to promise and emit result

const exampleTwo = source.mergeMap(val => myPromise(val));

//output: 'Hello World From Promise'

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

/\*

you can also supply a second argument which recieves the source value and emitted

value of inner observable or promise

\*/

const exampleThree = source

.mergeMap(val => myPromise(val),

(valueFromSource, valueFromPromise) => {

return `Source: ${valueFromSource}, Promise: ${valueFromPromise}`;

});

//output: "Source: Hello, Promise: Hello World From Promise!"

const subscribeThree = exampleThree.subscribe(val => console.log(val));

**partition**

#####signature: partition(predicate: function: boolean, thisArg: any): [Observable, Observable] *The gist: Split one observable into two based on predicate...*

([demo](http://jsbin.com/fuqojubaqu/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-partition))

const source = Rx.Observable.from([1,2,3,4,5,6]);

//first value is true, second false

const [evens, odds] = source.partition(val => val % 2 === 0);

/\*

Output:

"Even: 2"

"Even: 4"

"Even: 6"

"Odd: 1"

"Odd: 3"

"Odd: 5"

\*/

const subscribe = Rx.Observable.merge(

evens

.map(val => `Even: ${val}`),

odds

.map(val => `Odd: ${val}`)

).subscribe(val => console.log(val));

//if greater than 3 throw

const example = source

.map(val => {

if(val > 3){

throw `${val} greater than 3!`

}

return {success: val};

})

.catch(val => Rx.Observable.of({error: val}));

//split on success or error

const [success, error] = example.partition(res => res.success)

/\*

Output:

"Success! 1"

"Success! 2"

"Success! 3"

"Error! 4 greater than 3!"

\*/

const subscribeTwo = Rx.Observable.merge(

success.map(val => `Success! ${val.success}`),

error.map(val => `Error! ${val.error}`)

).subscribe(val => console.log(val));

**pluck**

#####signature: pluck(properties: ...args): Observable *The gist: Pick out nested properties...*

([demo](http://jsbin.com/netulokasu/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-pluck))

const source = Rx.Observable.from([

{name: 'Joe', age: 30},

{name: 'Sarah', age:35}

]);

//grab names

const example = source.pluck('name');

//output: "Joe", "Sarah"

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

const sourceTwo = Rx.Observable.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 exampleTwo = sourceTwo.pluck('job', 'title');

//output: "Developer" , undefined

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**publish**

#####signature: publish() : ConnectableObservable *The gist: Do nothing until connect is called, share source...*

([demo](http://jsbin.com/laguvecixi/edit?js,console) | [official docs](http://reactivex-rxjs5.surge.sh/function/index.html#static-function-publish))

//emit value every 1 second

const source = Rx.Observable.interval(1000);

const example = source

//side effects will be executed once

.do(() => 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)

**race**

#####signature: race(): Observable *The gist: Take the first observable to emit...*

([demo](http://jsbin.com/goqiwobeno/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-race))

//take the first observable to emit

const example = Rx.Observable.race(

//emit every 1.5s

Rx.Observable.interval(1500),

//emit every 1s

Rx.Observable.interval(1000).mapTo('1s won!'),

//emit every 2s

Rx.Observable.interval(2000),

//emit every 2.5s

Rx.Observable.interval(2500)

);

//output: "1s won!"..."1s won!"...etc

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

**repeat**

#####signature: repeat(scheduler: Scheduler, count: number): Observable *The gist: Repeat source specified number of times...*

([demo](http://jsbin.com/lexabovuqa/1/edit?js,console) | [official docs](http://reactivex-rxjs5.surge.sh/function/index.html#static-function-repeat))

//emit "Repeat this!"

const source = Rx.Observable.of('Repeat this!');

//repeat items emitted from source 3 times

const example = source.repeat(3);

//output: "Repeat this!"..."Repeat this!"..."Repeat this!"

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

//emit every second

const sourceTwo = Rx.Observable.interval(1000);

//take 5 values, repeat 2 times

const exampleTwo = sourceTwo.take(5).repeat(2);

//output: 0,1,2,3,4,5...0,1,2,3,4,5

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**retry**

#####signature: retry(number: number): Observable *The gist: Retry specified number of times on error...*

([demo](http://jsbin.com/yovacuxuqa/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-retry))

//emit value every 1s

const source = Rx.Observable.interval(1000);

const example = source

.flatMap(val => {

//throw error for demonstration

if(val > 5){

return Rx.Observable.throw('Error!');

}

return Rx.Observable.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!`)

});

**retryWhen**

#####signature: retryWhen(receives: notificationHandler, the: scheduler): Observable *The gist: Retry with additional logic...*

([demo](http://jsbin.com/miduqexalo/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-retryWhen))

//emit value every 1s

const source = Rx.Observable.interval(1000);

const example = source

.map(val => {

if(val > 5){

//error will be picked up by retryWhen

throw val;

}

return val;

})

.retryWhen(errors => errors

//log error message

.do(val => console.log(`Value ${val} was too high!`))

//restart in 5 seconds

.delayWhen(val => Rx.Observable.timer(val \* 1000))

);

/\*

output:

0

1

2

3

4

5

"Value 6 was too high!"

--Wait 5 seconds then repeat

\*/

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

**sample**

#####signature: sample(sampler: Observable): Observable *The gist: Sample from source when supplied observable emits...*

([demo](http://jsbin.com/wifaqipuse/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-sample))

//emit value every 1s

const source = Rx.Observable.interval(1000);

//sample last emitted value from source every 2s

const example = source.sample(Rx.Observable.interval(2000));

//output: 2..4..6..8..

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

const sourceTwo = Rx.Observable.zip(

//emit 'Joe', 'Frank' and 'Bob' in sequence

Rx.Observable.from(['Joe', 'Frank', 'Bob']),

//emit value every 2s

Rx.Observable.interval(2000)

);

//sample last emitted value from source every 2.5s

const exampleTwo = sourceTwo.sample(Rx.Observable.interval(2500));

//output: ["Joe", 0]...["Frank", 1]...........

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**scan**

#####signature: scan(accumulator: function, seed: any): Observable *The gist: Reduce over time...*

([demo](http://jsbin.com/jopikihuvu/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-scan))

const testSubject = new Rx.Subject();

//basic scan example, sum over time starting with zero

const basicScan = testSubject

.startWith(0)

.scan((acc, curr) => acc + curr);

//log accumulated values

const subscribe = basicScan.subscribe(val => console.log('Accumulated total:', val));

//next values into subject, adding to the current sum

testSubject.next(1); //1

testSubject.next(2); //3

testSubject.next(3); //6

const testSubjectTwo = new Rx.Subject();

//scan example building an object over time

const objectScan = testSubjectTwo.scan((acc, curr) => Object.assign({}, acc, curr), {});

//log accumulated values

const subscribe = objectScan.subscribe(val => console.log('Accumulated object:', val));

//next values into subject, adding properties to object

testSubjectTwo.next({name: 'Joe'}); // {name: 'Joe'}

testSubjectTwo.next({age: 30}); // {name: 'Joe', age: 30}

testSubjectTwo.next({favoriteLanguage: 'JavaScript'}); // {name: 'Joe', age: 30, favoriteLanguage: 'JavaScript'}

**share**

#####signature: share(): Observable *The gist: Share observable among multiple subscribers...*

([demo](http://jsbin.com/jobiyomari/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-share))

//emit value in 1s

const source = Rx.Observable.timer(1000);

//log side effect, emit result

const example = source

.do(() => 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.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));

**single**

#####signature: single(a: Function): Observable *The gist: Emit single item that matches condition...*

([demo](http://jsbin.com/solecibuza/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-single))

//emit (1,2,3,4,5)

const source = Rx.Observable.from([1,2,3,4,5]);

//emit one item that matches predicate

const example = source.single(val => val === 4);

//output: 4

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

**skip**

#####signature: skip(the: Number): Observable *The gist: Skip a specified number of emitted items...*

([demo](http://jsbin.com/hacepudabi/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-skip))

//emit every 1s

const source = Rx.Observable.interval(1000);

//skip the first 5 emitted values

const example = source.skip(5);

//output: 5...6...7...8........

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

**skipUntil**

#####signature: skipUntil(the: Observable): Observable *The gist: Skip emitted items from source until inner observable emits...*

([demo](http://jsbin.com/tapizososu/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-skipUntil))

//emit every 1s

const source = Rx.Observable.interval(1000);

//skip emitted values from source until inner observable emits (6s)

const example = source.skipUntil(Rx.Observable.timer(6000));

//output: 5...6...7...8........

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

**skipWhile**

#####signature: skipWhile(predicate: Function): Observable *The gist: Skip emitted items from source until provided expression is false...*

([demo](http://jsbin.com/bemikuleya/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-skipWhile))

//emit every 1s

const source = Rx.Observable.interval(1000);

//skip emitted values from source while value is less than 5

const example = source.skipWhile(val => val < 5);

//output: 5...6...7...8........

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

**startWith**

#####signature: startWith(an: Values): Observable *The gist: Emit specified item first...*

([demo](http://jsbin.com/jeyakemume/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-startWith))

//emit (1,2,3)

const source = Rx.Observable.of(1,2,3);

//start with 0

const example = source.startWith(0);

//output: 0,1,2,3

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

//emit ('World!', 'Goodbye', 'World!')

const sourceTwo = Rx.Observable.of('World!', 'Goodbye', 'World!');

//start with 'Hello', concat current string to previous

const exampleTwo = sourceTwo

.startWith('Hello')

.scan((acc, curr) => `${acc} ${curr}`);

/\*

output:

"Hello"

"Hello World!"

"Hello World! Goodbye"

"Hello World! Goodbye World!"

\*/

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**switchMap**

#####signature: switchMap(a: Observable): Observable *The gist: When source emits, switch to and emit values emitted from latest inner observable*

([demo](http://jsbin.com/decinatisu/1/edit?js,console,output) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-switchMap))

//emit immediately, then every 5s

const source = Rx.Observable.timer(0, 5000);

//switch to new inner observable when source emits, emit items that are emitted

const example = source.switchMap(() => Rx.Observable.interval(500));

//output: 0,1,2,3,4,5,6,7,8,9...0,1,2,3,4,5,6,7,8

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

//emit every click

const sourceTwo = Rx.Observable.fromEvent(document, 'click');

//if another click comes within 3s, message will not be emitted

const exampleTwo = sourceTwo.switchMap(val => Rx.Observable.interval(3000).mapTo('Hello, I made it!'));

//(click)...3s...'Hello I made it!'...(click)...2s(click)...

const subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**window**

#####signature: window(windowBoundaries: Observable): Observable *The gist: Observable of values for window of time*

([demo](http://jsbin.com/jituvajeri/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-window))

//emit immediately then every 1s

const source = Rx.Observable.timer(0, 1000);

const example = source

.window(Rx.Observable.interval(3000))

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

**windowCount**

#####signature: windowCount(windowSize: number, startWindowEvery: number): Observable *The gist: Observable of values from source, emitted each time count is fulfilled*

([demo](http://jsbin.com/nezuvacexe/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-windowCount))

//emit every 1s

const source = Rx.Observable.interval(1000);

const example = source

//start new window every 4 emitted values

.windowCount(4)

.do(() => console.log('NEW WINDOW!'))

const subscribeTwo = example

//window emits nested observable

.mergeAll()

/\*

output:

"NEW WINDOW!"

0

1

2

3

"NEW WINDOW!"

4

5

6

7

\*/

.subscribe(val => console.log(val));

**windowTime**

#####signature: windowTime(windowTimeSpan: number, windowCreationInterval: number, scheduler: Scheduler): Observable *The gist: Emit an observable of values collected from source every specified time span*

([demo](http://jsbin.com/mifayacoqo/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-windowTime))

//emit immediately then every 1s

const source = Rx.Observable.timer(0,1000);

const example = source

//start new window every 3s

.windowTime(3000)

.do(() => console.log('NEW WINDOW!'))

const subscribeTwo = example

//window emits nested observable

.mergeAll()

/\*

output:

"NEW WINDOW!"

0

1

2

"NEW WINDOW!"

3

4

5

\*/

.subscribe(val => console.log(val));

**windowToggle**

#####signature: windowToggle(openings: Observable, closingSelector: function(value): Observable): Observable *The gist: Collect and emit observable of values from source between opening and closing emission*

([demo](http://jsbin.com/xasofupuka/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-windowToggle))

//emit immediately then every 1s

const source = Rx.Observable.timer(0,1000);

//toggle window on every 5

const toggle = Rx.Observable.interval(5000);

const example = source

//turn window on every 5s

.windowToggle(toggle, (val) => Rx.Observable.interval(val \* 1000))

.do(() => console.log('NEW WINDOW!'))

const subscribeTwo = example

//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));

**windowWhen**

#####signature: windowWhen(closingSelector: function(): Observable): Observable *The gist: Close window at specified time frame emitting observable of collected values from source, repeat...*

([demo](http://jsbin.com/tuhaposemo/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-windowWhen))

//emit immediately then every 1s

const source = Rx.Observable.timer(0,1000);

const example = source

//close window every 5s and emit observable of collected values from source

.windowWhen((val) => Rx.Observable.interval(5000))

.do(() => console.log('NEW WINDOW!'))

const subscribeTwo = example

//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));

**withLatestFrom**

#####signature: withLatestFrom(other: Observable, project: Function): Observable *The gist: When source emits, also give last value emitted from another observable...*

([demo](http://jsbin.com/xehucaketu/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#instance-method-withLatestFrom))

//emit every 5s

const source = Rx.Observable.interval(5000);

//emit every 1s

const secondSource = Rx.Observable.interval(1000);

const example = source

.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));

//withLatestFrom slower than source

const exampleTwo = secondSource

//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 subscribeTwo = exampleTwo.subscribe(val => console.log(val));

**zip**

#####signature: zip(observables: \*): Observable *The gist: After all observables emit, emit values as an array...*

([demo](http://jsbin.com/torusemimi/1/edit?js,console) | [official docs](http://reactivex.io/rxjs/class/es6/Observable.js~Observable.html#static-method-zip))

const sourceOne = Rx.Observable.of('Hello');

const sourceTwo = Rx.Observable.of('World!');

const sourceThree = Rx.Observable.of('Goodbye');

const sourceFour = Rx.Observable.of('World!');

//wait until all observables have emitted a value then emit all as an array

const example = Rx.Observable

.zip(

sourceOne,

sourceTwo.delay(1000),

sourceThree.delay(2000),

sourceFour.delay(3000)

);

//output: ["Hello", "World!", "Goodbye", "World!"]

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

//emit every 1s

const interval = Rx.Observable.interval(1000);

//when one observable completes no more values will be emitted

const exampleTwo = Rx.Observable

.zip(

interval,

interval.take(2)

);

//output: [0,0]...[1,1]

const subscribe = exampleTwo.subscribe(val => console.log(val));