



SUBJECT TYPES

	Subject	Relay
Publish	No state, errors	No state, no errors
Behavior	State, errors	State, no errors

UIKIT EXTENSIONS

```
1 let button: UIButton = UIButton(frame: .zero)
2 button.rx.tap // ControlEvent<Void>
3
4 let control: UIControl = UIControl(frame: .zero)
5 control.rx.controlEvent(_ controlEvents: UIControl.Event) // ControlEvent<Void>
6
7 let segmentedControl: UISegmentedControl = UISegmentedControl(frame: .zero)
8 segmentedControl.rx.selectedSegmentIndex // ControlProperty<Int>
9
10 let textField: UITextField = UITextField(frame: .zero)
11 textField.rx.text // ControlProperty<String?>
12
13 let tableView: UITableView = UITableView(frame: .zero, style: .plain)
14 tableView.rx.itemSelected // ControlEvent<IndexPath>
15 tableView.rx.modelSelected(_ modelType: T.Type) // ControlEvent<T>
16 tableView.rx.items(cellIdentifier: String) // (0) -> (@escaping (Int, S.Iterator.Element, Cell) -> Void) ->
17 Disposable where S : Sequence, S == O.E, Cell : UITableViewCell, O : ObservableType
18
19 let notificationCenter: NotificationCenter = NotificationCenter.default
20 notificationCenter.rx.notification(_ name: Notification.Name?) // Observable<Notification>
```

MORE OPERATORS

- List: <http://reactivex.io/documentation/operators.html>
- Visual aid: <http://rxmarbles.com>

BASIC INFORMATION

Observable sequences can emit 0 or more events over their lifetimes. 3 types of events:

- .next(value: T) – new value from the observable
- .error(error: Error) – observable encountered the error; terminates the sequence
- .completed – observable finished emitting events; terminates the sequence

CREATING OBSERVABLES

asObservable - convert various objects into Observables

```
1 let field = UITextField(frame: .zero)
2 field.rx.text.orEmpty.asObservable()
```

create - create an Observable from scratch

```
1 Observable<String>.create { subscribe in
2     subscribe.onNext("Rx Workshops are cool")
3     subscribe.onCompleted()
4
5     return Disposables.create()
6 }
```

from(_: [T]) - convert an array into Observable

```
1 Observable<Int>.from([1, 2, 3])
2     .subscribe { print($0) }
```

```
next(1)
next(2)
next(3)
completed
```

of - convert values into Observable

```
1 Observable<Int>.of(1, 2, 3)
2     .subscribe { print($0) }
```

```
next(1)
next(2)
next(3)
completed
```

empty - emit no items; terminates normally

error - emit no items; terminates with an error

```
1 Observable<String>.error(RxError.unknown)
2     .subscribe { print($0) }
```

```
error(Unknown error occurred.)
```

never - emit no items; does not terminate

just - emit a particular item

```
1 Observable<String>.just("Hello Rx!")
2     .subscribe { print($0) }
```

```
next(Hello Rx!)
completed
```

TRANSFORMING OBSERVABLES

map

```
1 Observable<Int>.of(3, 4, 5)
2     .map { "Mambo no. \($0)" }
3     .subscribe { print($0) }
```

```
next(Mambo no. 3)
next(Mambo no. 4)
next(Mambo no. 5)
completed
```

flatMap

```
1 func fetchUser(id: Int) -> Observable<User> {
2     return URLSession.shared.rx // ...
3 }
4
5 Observable<Int>.of(1, 2, 3)
6     .flatMap { fetchUser(id: $0) }
7     .subscribe { print($0) }
```

```
next(User(id: 1))
next(User(id: 2))
next(User(id: 3))
completed
```



TRANSFORMING OBSERVABLES

flatMapFirst

```
1 Observable<Int>.of(1, 2, 3)
2   .flatMapFirst { fetchUser(id: $0) }
3   .subscribe { print($0) }
```

```
next(User(id: 1))
completed
```

flatMapLatest

```
1 Observable<Int>.of(1, 2, 3)
2   .flatMapLatest { fetchUser(id: $0) }
3   .subscribe { print($0) }
```

```
next(User(id: 3))
completed
```

FILTERING OBSERVABLE

filter

```
1 Observable<Int>.of(1, 2, 3, 4, 5, 6)
2   .filter { $0 > 4 }
3   .subscribe { print($0) }
```

```
next(5)
next(6)
completed
```

distinctUntilChanged

```
1 Observable<Int>.of(1, 1, 1, 3, 3, 5, 1, 5)
2   .distinctUntilChanged()
3   .subscribe { print($0) }
```

```
next(1)
next(3)
next(5)
next(1)
next(5)
completed
```

skip(_: Int)

```
1 Observable<Int>.of(200, 300, 400, 500, 600)
2   .skip(3)
3   .subscribe { print($0) }
```

```
next(500)
next(600)
completed
```

CONDITIONAL AND BOOLEAN OPERATORS

takeWhile

```
1 Observable<Int>.of(1, 2, 3, 4, 5, 6)
2   .takeWhile { $0 < 4 }
3   .subscribe { print($0) }
```

```
next(1)
next(2)
next(3)
completed
```

ERROR HANDLING OPERATORS

catchError

```
1 Observable<String>.error(RxError.unknown)
2   .catchError({ error -> Observable<String> in
3     return .just("RxSwift")
4   })
5   .subscribe { print($0) }
```

```
next(RxSwift)
completed
```

catchErrorJustReturn

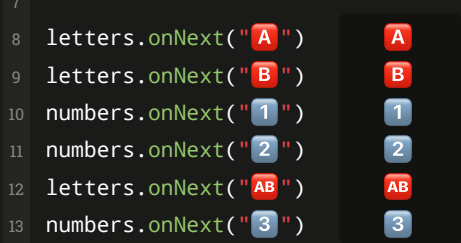
```
1 Observable<String>.error(RxError.unknown)
2   .catchErrorJustReturn("RxSwift")
3   .subscribe { print($0) }
```

```
next(RxSwift)
completed
```

COMBINING OBSERVABLES

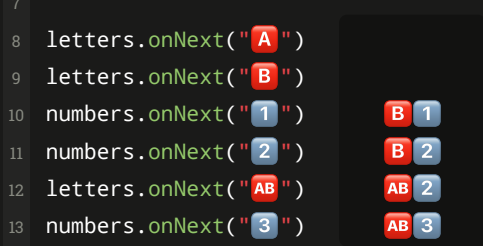
merge

```
1 let letters = PublishSubject<String>()
2 let numbers = PublishSubject<String>()
3
4 Observable
5   .merge(letters, numbers)
6   .subscribe(onNext: { print($0) })
7
```



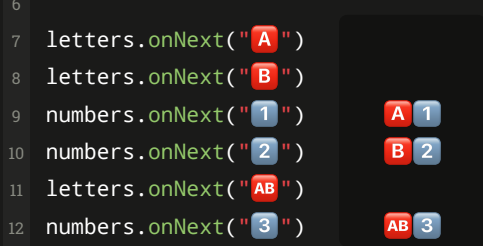
combineLatest

```
1 let letters = PublishSubject<String>()
2 let numbers = PublishSubject<String>()
3
4 Observable
5   .combineLatest(letters, numbers) { "\( $0) \( $1)" }
6   .subscribe(onNext: { print($0) })
7
```



zip

```
1 let letters = PublishSubject<String>()
2 let numbers = PublishSubject<String>()
3
4 Observable.zip(letters, numbers) { "\( $0) \( $1)" }
5   .subscribe(onNext: { print($0) })
6
```



AGGREGATE OPERATORS

reduce

```
1 Observable<Int>.of(1, 2, 3, 4, 5)
2   .reduce("") { $0.appending("\( $1)") }
3   .subscribe { print($0) }
```

```
next(12345)
completed
```

toArray

```
1 Observable.of(1, 2, 3, 4, 5)
2   .toArray()
3   .subscribe { print($0) }
```

```
next([1, 2, 3, 4, 5])
completed
```

LIFECYCLE

Cold observable

```
1 let coldObservable = Observable.just("Hello Rx!")
2 let subscription = coldObservable
3   .subscribe { print($0) }
4   .disposed(by: disposeBag)
5
6 // Subscription will be disposed after completed
7 // event.
```

Hot observable

```
1 let hotObservable = button.rx.tap.asObservable()
2 let subscription = hotObservable
3   .subscribe { print($0) }
4   .disposed(by: disposeBag)
5
6 // Subscription will be disposed when disposeBag
7 // will be deallocated.
```