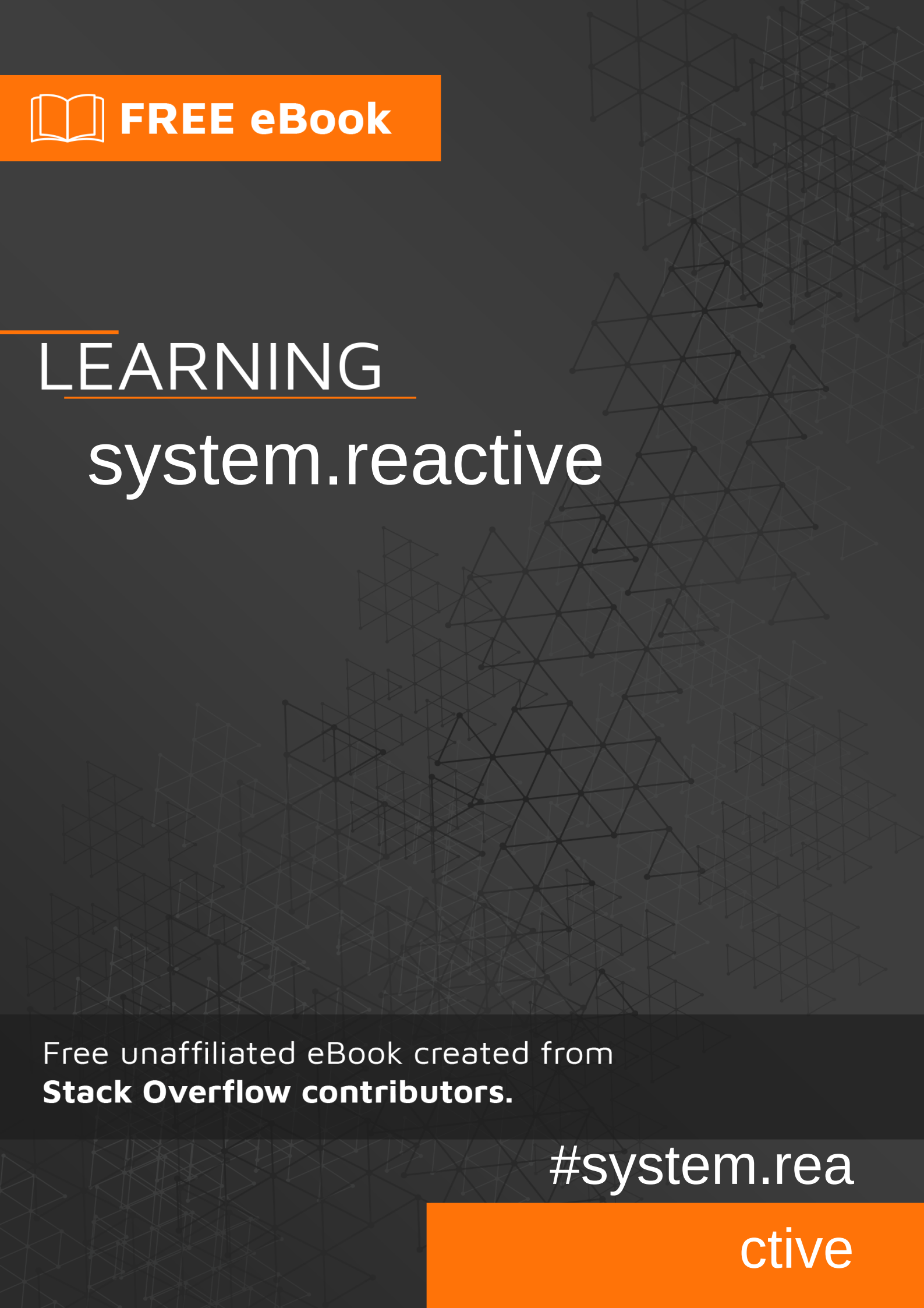
FREE eBook
LEARNING
system.reactive
Free unaffiliated eBook created from
Stack Overflow contributors.
#system.rea
ctive

Table of Contents

About 1

Chapter 1: Getting started with system.reactive 2

Remarks 2

Examples 2

Using Rx in your project 2

Filtering the values of an observable 2

Selecting a new value for each value in an observable 2

Subscribing/unsubscribing to an observable (IDisposable) 2

Subscribing to an observable (CancellationToken) 2

Wrapping an async method as an observable 3

Sharing a single subscription (Publish) 3

Sharing a single subscription (Publish + RefCount) 3

Installation or Setup 3

Throttling a stream 4

Ignoring repeated values 5

Get a running aggregation 5

Credits 7

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: system-reactive

It is an unofficial and free system.reactive ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official system.reactive.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with system.reactive

Remarks

This section provides an overview of what system.reactive is, and why a developer might want to use it.

It should also mention any large subjects within system.reactive, and link out to the related topics. Since the Documentation for system.reactive is new, you may need to create initial versions of those related topics.

Examples

Using Rx in your project

Install the NuGet package `System.Reactive`, then add this using statement to access the Rx extension methods:

```
using System.Reactive.Linq;
```

Filtering the values of an observable

```
emails.Where(email => email.From == "John")
```

Selecting a new value for each value in an observable

```
emails.Select(email => email.Body)
```

Subscribing/unsubscribing to an observable (IDisposable)

Subscription returns an `IDisposable`:

```
IDisposable subscription = emails.Subscribe(email =>
    Console.WriteLine("Email from {0} to {1}", email.From, email.To));
```

When you are ready to unsubscribe, simply dispose the subscription:

```
subscription.Dispose();
```

Subscribing to an observable (CancellationToken)

```
emails.Subscribe(email =>
    Console.WriteLine("Email from {0} to {1}", email.From, email.To),
    cancellationToken);
```

Wrapping an async method as an observable

Given an `async` method like this:

```
Task<string> GetNameAsync(CancellationToken cancellationToken)
```

Wrap it as an `IObservable<string>` like this:

```
Observable.FromAsync(cancellationToken => GetNameAsync(cancellationToken))
```

Sharing a single subscription (Publish)

Given an `IObservable<Offer>` of `offers` from merchants to buy or sell some type of item at a fixed price, we can match pairs of buyers and sellers as follows:

```
var sellers = offers.Where(offer => offer.IsSell).Select(offer => offer.Merchant);
var buyers = offers.Where(offer => offer.IsBuy).Select(offer => offer.Merchant);
var trades = Observable.Zip(sellers, buyers, (seller, buyer) => new Trade(seller, buyer));
```

The problem with this is that each subscription to `trades` will subscribe to `offers` twice. We can make `sellers` and `buyers` share a single subscription to `offers` by using `Publish`:

```
var trades = offers.Publish(_offers =>
{
    var sellers = _offers.Where(offer => offer.IsSell).Select(offer => offer.User);
    var buyers = _offers.Where(offer => offer.IsBuy).Select(offer => offer.User);
    return Observable.Zip(sellers, buyers, (seller, buyer) => new Trade(seller, buyer));
});
```

Sharing a single subscription (Publish + RefCount)

This code will subscribe to the `emails` observable twice:

```
emails.Where(email => email.From == "John").Subscribe(email => Console.WriteLine("A"));
emails.Where(email => email.From == "Mary").Subscribe(email => Console.WriteLine("B"));
```

To share a single subscription to `emails`, we can use `Publish` and `RefCount` instead:

```
var _emails = emails.Publish().RefCount();
_emails.Where(email => email.From == "John").Subscribe(email => Console.WriteLine("A"));
_emails.Where(email => email.From == "Mary").Subscribe(email => Console.WriteLine("B"));
```

Installation or Setup

Reactive Extensions are published on both NuGet and MyGet.

Installing and using them is therefore the same as any other NuGet package:

```
Install-Package System.Reactive
```

NB package names changed between v2 and v3. See the README on Github for more info

> Breaking changes
>
> The NuGet packages have changed their package naming in the move from v2.x.x to >v3.0.0
>
> Rx-Main is now System.Reactive Rx-Core is now System.Reactive.Core Rx-Interfaces is now System.Reactive.Interfaces Rx-Linq is now System.Reactive.Linq Rx-PlatformServices is now System.Reactive.PlatformServices Rx-Testing is now Microsoft.Reactive.Testing

Throttling a stream

Let's say you need to implement an automatic search box, but the search operation is somewhat costly, like sending a web request or hitting up a database. You may want to limit the amount of search being done.

For example, the user is typing "C# Reactive Extensions" in the search box :

```
IObservable<string> TypingSearchText()
{
    return Observable.Create<string>(o =>
    {
        const string SearchText = "C# Reactive Extensions";

        var builder = new StringBuilder();
        foreach (var c in SearchText)
        {
            builder.Append(c);

            // notify that the search text has been changed
            o.OnNext(builder.ToString());

            // pause between each character to simulate actual typing
            Thread.Sleep(125);

            // spent some time to think about the next word to type
            if (c == ' ')
                Thread.Sleep(1000);
        }

        o.OnCompleted();

        return () => { /* nothing to dispose here */ };
    });
}
```

Now, we don't want to perform the search every time the user presses a key. Instead, it will be done whenever the user stops typing longer than half a second :

```
TypingSearchText()
    // print the changes
    .Do(x => Console.WriteLine("Typing: " + x))

    // ignore changes that happens within 500ms of each other
    .Throttle(TimeSpan.FromMilliseconds(500))

    // some costly operation
    .Subscribe(x => Console.WriteLine("Searching: " + x));
```

Output :

```
Typing: C
Typing: C#
Typing: C#
Searching: C#
Typing: C# R
Typing: C# Re
...
Typing: C# Reactive
Typing: C# Reactive
Searching: C# Reactive
Typing: C# Reactive E
Typing: C# Reactive Ex
...
Typing: C# Reactive Extension
Typing: C# Reactive Extensions
Searching: C# Reactive Extensions
```

Ignoring repeated values

There are two operators for filtering duplicates:

```
emails.Distinct(); // Never see the same value twice
emails.DistinctUntilChanged(); // Never see the same value twice in a row
```

You can also pass in a predicate:

```
emails.DistinctUntilChanged(x => x.Length); // Never see the same length email twice in a row
```

Get a running aggregation

Suppose you have a hot observable for which you would love to keep the count of. It could be the `IObservable<StockTick>` and you want to keep count of the average trade volume. You can use `Scan` for that.

```
var tradeVolume = stockTicks.Select(e => e.Price)
    .Scan(0.0m, (aggregated, newtick) => aggregated + newtick)
    .Select((aggregated, index) => aggregated / (index + 1))
```

Now you can simply subscribe to your trade volume which is live updated upon receipt of every new Tick.

```
var subscription = tradeVolume.Subscribe(vol => Console.WriteLine("New trade volume is {0}",
vol);
```

Read Getting started with system.reactive online: https://riptutorial.com/system-reactive/topic/1325/getting-started-with-system-reactive

Credits

S. No	Chapters	Contributors
1	Getting started with system.reactive	Benjol, Community, Dorus, fahadash, Michael Richardson, Paul D'Ambra, sdgfsdh, Shlomo, Timothy Shields, Xiaoy312