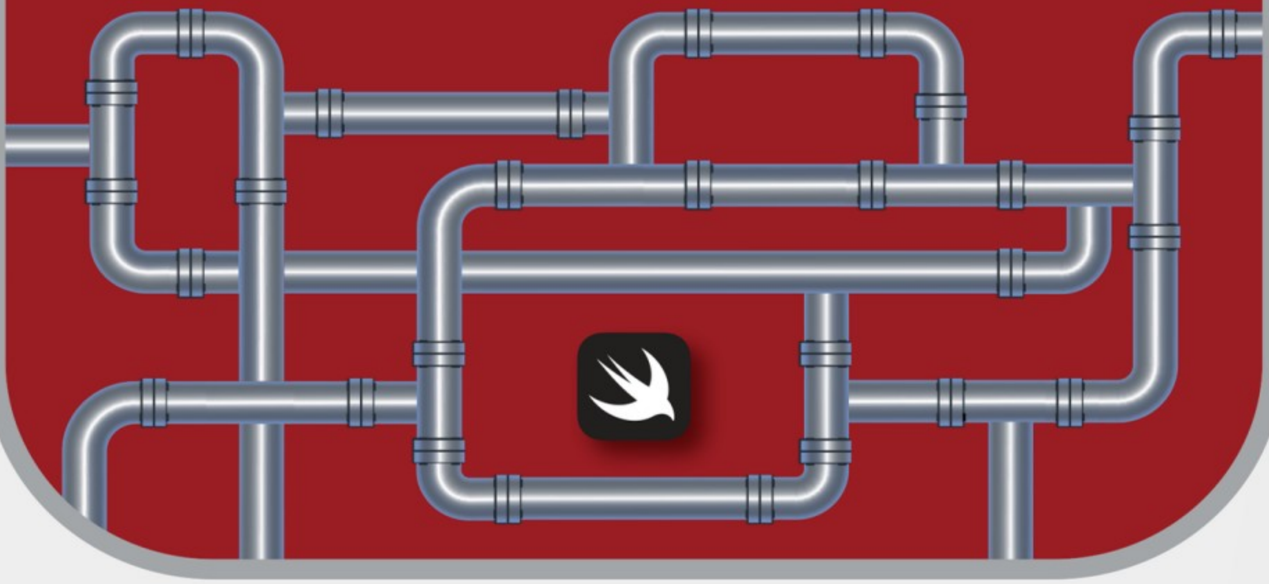
A Combine Kickstart

DANIEL H STEINBERG



Introducing the Declarative Framework for Processing Values over Time

Editors Cut

A Combine Kickstart Excerpt

Introducing The Declarative Framework

For Processing Values Over Time

by Daniel H Steinberg

Editors Cut

Copyright

ISBN-13: 978-1-944994-02-0

Book Version

This is version 0.4 for Swift 5.3 (tested and verified on Swift 5.5), Xcode 12.4 (tested and verified in Xcode 13), macOS Big Sur and Monterey, and iOS 14 and iOS 15 released August 2021. All code has been tested on Apple Silicon.

Code Download

Visit https://github.com/editorscut/ec011CombineKickstart for all of the code for this book.

Run it in Xcode 12 or higher. All code is written in Swift.

Recommended Settings

The ePub is best viewed in scrolling mode on an iPad. On smaller devices I also choose landscape. For some reason that I don't understand, scrolling mode is supported by Apple's Books app on the iPad but not on the Mac. If you view this book in Apple's Books app, choose "Let lines break naturally" in Preferences > General.

Submit Errata

Submit your errata here for the book or for the source code by selecting New Issue. Please provide the book version listed above, chapter, section, and page number in your issue so that I can find it and, if possible, resolve it quickly.

Official Links

Please check http://developer.apple.com for additional resources including videos, sample code, documentation, and forums. You'll also find information on what is required to take advantage of these resources.

Apple has posted videos, slides, and sample code from the Worldwide Developers Conference.

Legal

Table Of Contents

Chapter 3 - Section 7: Custom Publisher Properties

Custom Publisher Properties

In this section we create a custom publisher from an existing private publisher and store the publisher in a public property.

This may be one of my favorite techniques.

We will keep our private `@Published model` in `State` and use it as part of a public publisher of `Int`s that we'll add to `State`.

It's important that you keep these two ideas separate in your head:

- the publisher or pipeline for values, and
- the values that are published and flow through the pipes.

Cleaning up

Once again, we begin by cleaning up. Remove the contents of `contentsSubscription()` in `Link`.

03/07/Forest/Forest/Link.swift

```
extension Link {
  func contentsSubscription() {
  }
}
```

Remove `ObservableObject` and `value` from `State`.

03/07/Forest/FarFarAway/State.swift

```
import Combine

public class State~~: ObservableObject~~ {
  @Published private var model = Model()

  public init() {}
}

extension State {
  ~~public var value: Int {~~
    ~~model.value~~
  ~~}~~

  public func next() {
    model = model.next
  }
}
```

We're going to create a public property named `valuePublisher` that is a publisher of `Int`s that never fails.

Value Publisher

Create a new property named `valuePublisher` in `State`.

03/07/Forest/FarFarAway/State.swift

```
public class State {
  @Published private var model = Model()

  public let valuePublisher =
  // more to come

  public init() {}
}
```

`valuePublisher` will start with the `$model` publisher.

Experienced Swift developers know that if one property depends on another the dependent property must be declared to be `lazy`. All `lazy` properties must be `var`s. I'm cautious enough that if I have a `public var` I mark it as `private(set)` so others can read it but not alter it.

03/07/Forest/FarFarAway/State.swift

```
public class State {
  @Published private var model = Model()

  lazy public private(set) var valuePublisher
    = $model
    // still working on it
  public init() {}
}
```

This may not yet build as `Model` is `internal`.

Option-Click `valuePublisher` and you'll see the type is

```
Published<Model>.Publisher
```

This is the same as the type of the publisher `$model`.

Customizing the Output

We'd like `valuePublisher` to publish `Int`'s representing `model.value` and we don't want to send the initial value `0`.

You've performed both tasks before. Give it a try here.

Here's what I've done:

03/07/Forest/FarFarAway/State.swift

```
public class State {
  @Published private var model = Model()

  lazy public private(set) var valuePublisher
    = $model
    .dropFirst()
    .map(\.value)

  public init() {}
}
```

Our `valuePublisher` does everything we want it to do but Option-Click it to see that it has this unfortunate type:

```
Publishers.MapKeyPath<Publishers.Drop<Published<Model>.Publisher>,
                      Int>
```

The type let's us know that it begins with a publisher of `Model`s, drops one or more of them, then maps `Model` to `Int`.

This type does not make it easy for potential subscribers know that in the end it is getting a publisher of `Int`s that never fails. It also is exposing `Model` which is an internal detail as is the fact that we used `Drop`, `Map`, and `Published` publishers.

We should clean up our public facing interface.

Erase to any Publisher

Our goal is to expose `valuePublisher` to `Link` and others outside of the *FarFarAway* module.

They shouldn't care that we used `dropFirst()` and `map()`. All they care is that they're getting a publisher of `Int`s that never fails.

We use type erasure to communicate that fact and to keep others from depending on implementation details that we may later change.

Declare `valuePublisher` to be `AnyPublisher<Int, Never>`.

03/07/Forest/FarFarAway/State.swift

```
public class State {
  @Published private var model = Model()

  lazy public private(set) var valuePublisher:
                                  AnyPublisher<Int, Never>
    = $model
    .dropFirst()
    .map(\.value)  // we've introduced an error

  public init() {}
}
```

Build *FarFarAway* and you can see that we've introduced an error that appears at `map(\.value)`.

Our error is that there's a type mismatch. We're told that the return type of `valuePublisher` is `AnyPublisher<Int, Never>` but we're returning something of type `Publishers.MapKeyPath<Publishers.Drop<Published<Model>.Publisher>, Int>`.

We fix this with the method `eraseToAnyPublisher` which erases this complex type and replaces it with `AnyPublisher` with the correct `Output` and `Failure` types.

03/07/Forest/FarFarAway/State.swift

```
public class State {
  @Published private var model = Model()

  lazy public private(set) var valuePublisher:
                                AnyPublisher<Int, Never>
    = $model
    .dropFirst()
    .map(\.value)
    .eraseToAnyPublisher()

  public init() {}
}
```

Now this is fit to be consumed.

Subscribing to Value Publisher

State has a public publisher of Ints that never fails so we can easily connect to it in Link.

03/07/Forest/Forest/Link.swift

```
extension Link {
  func contentsSubscription() {
    state.valuePublisher              // Pub<Int, Never>
      .assignDescription(asOptionalTo: &$contents)
  }
}
```

We've dramatically simplified contentsSubscription().

So...

I think we've created an incredibly clear and clean publisher chain in three pieces.

The first piece in State takes $model and publishes an Int representing any new values. It also is where the decision to drop the first element is made. This publisher doesn't care about initial values, it is only publishing changes.

The second piece in Link prepares each Int it receives in valuePublisher for display by placing it in a String? that is republished.

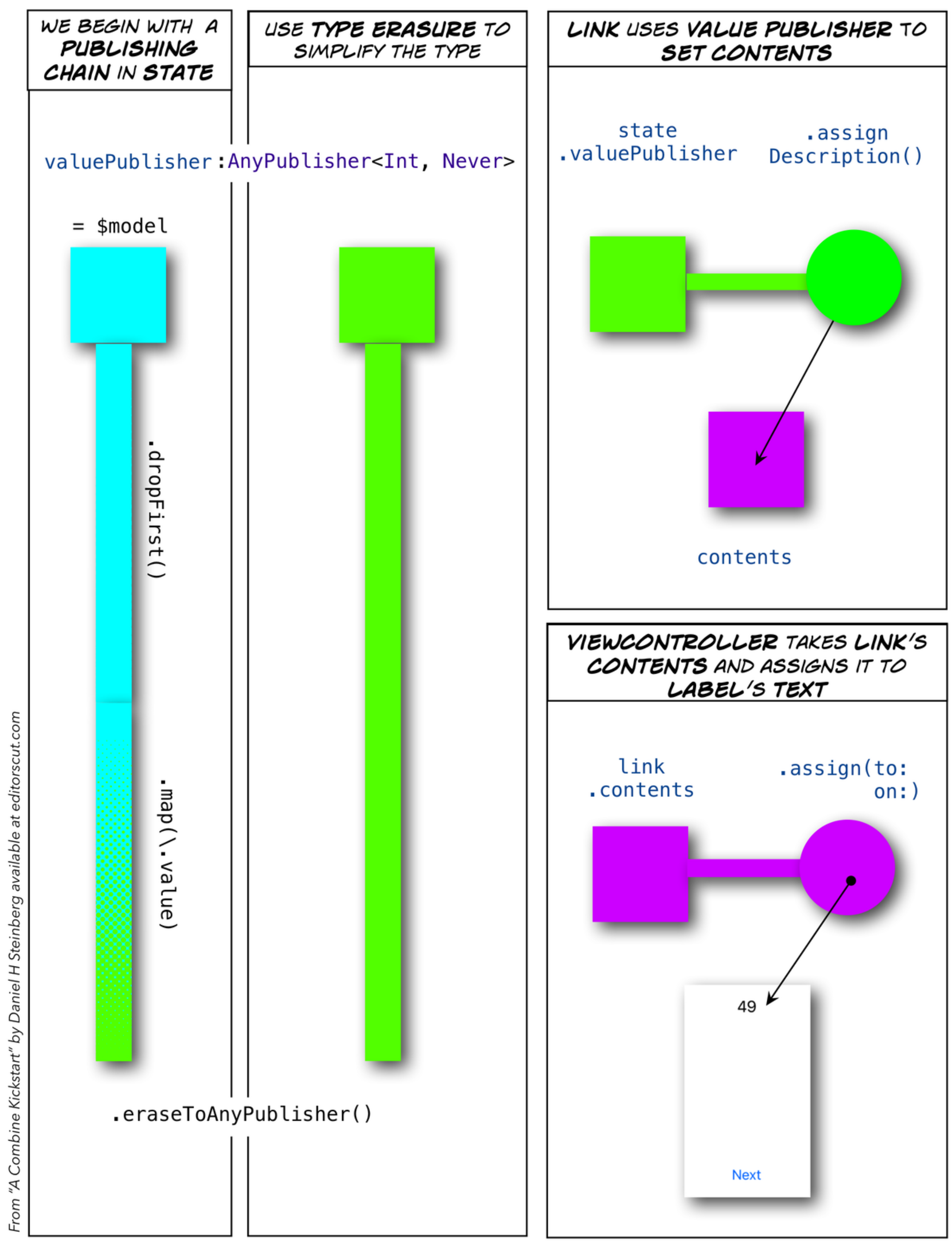
The third piece in ViewController takes each prepared String? it receives from $contents and displays it in label's text.

In the next section we replace our UIKit code with SwiftUI.

Before we do, I want to point out that we have made all of our changes in `Link` and `State`. We haven't touched the model or the UI.

We end this section with a visual summary of what we've done.

Custom publishers


WE BEGIN WITH A PUBLISHING CHAIN IN STATE
USE TYPE ERASURE TO SIMPLIFY THE TYPE
LINK USES VALUE PUBLISHER TO SET CONTENTS
valuePublisher:AnyPublisher<Int, Never>
= $model
.dropFirst()
.map(\.value)
.eraseToAnyPublisher()
state .valuePublisher
.assign Description()
contents
VIEWCONTROLLER TAKES LINK'S CONTENTS AND ASSIGNS IT TO LABEL'S TEXT
link .contents
.assign(to: on:)
49
Next
From "A Combine Kickstart" by Daniel H Steinberg available at editorscut.com