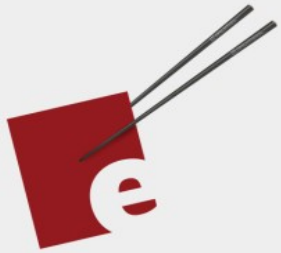
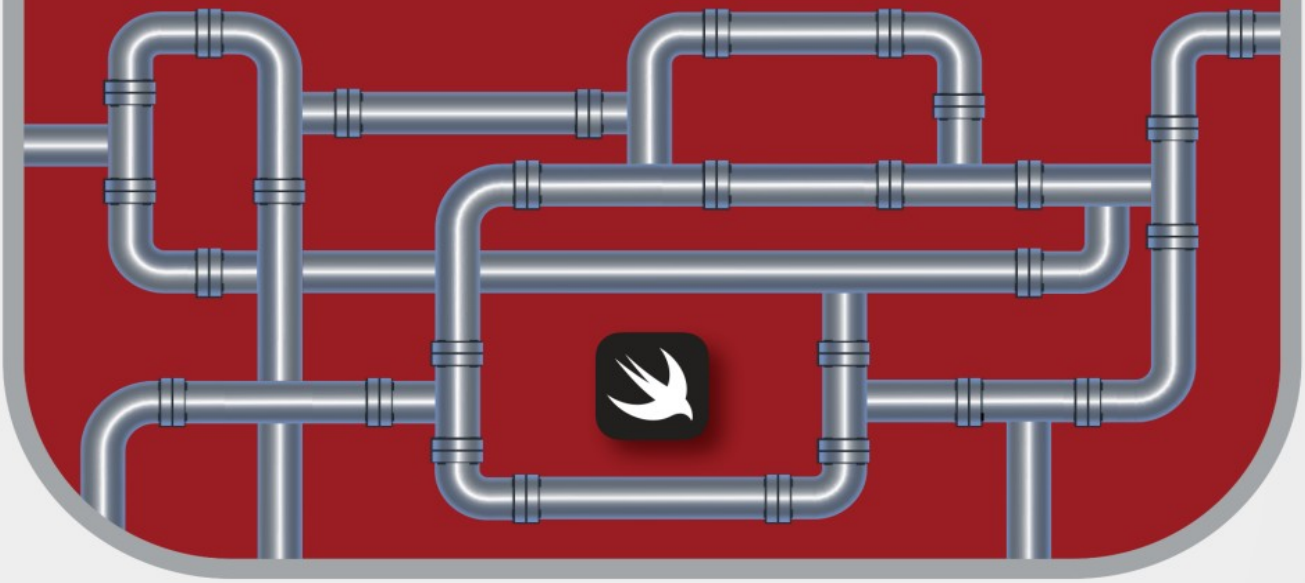


A Combine Kickstart

DANIEL H STEINBERG



Introducing the Declarative Framework for Processing Values over Time

Editors Cut

A Combine Kickstart

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 0.3 for Swift 5.3, Xcode 12.4, macOS Big Sur, and iOS 14 released February 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 1

The Forest

Sections:

My goal in this short book it to give you a feel for what problems the Combine framework helps you solve and best practices in using it.

I assume that you are an experienced developer who is familiar with how components communicate in iOS and macOS apps. You've worked with target-action, delegates, closures, and perhaps notifications, URLSessions, and other familiar mechanisms for asynchronous communication.

Combine unifies and, in my opinion, simplifies this family into a single API.

There's a ton to learn and it's easy to get lost in all of the details of the Combine Framework.

In this chapter we begin with an example that demonstrates how the various pieces of Combine work together. I just want to give you an overview to help you build a mental model. We're not going to begin with definitions and theory.

We'll cover definitions and theory later after you've had a chance to play with Combine a bit. Over time you'll become more and more powerful.

There are an initially overwhelming number of different methods, classes, and structs that are part of the Combine framework. We are not going exhaustively cover them all.

On the other hand, we do cover a surprising number of them.

My goal is that when we're done, you'll be able to scan through the docs and find the piece that does what you need or, if it doesn't exist, build it yourself.

We're going to work some examples and you'll get an idea of what Combine is and how to use it to great advantage.

When we get to the end you'll understand important details of the individual parts but more importantly, you'll have the big picture of how they fit together.

"But Daniel," you complain, "that doesn't say anything about what Combine is or does."

It's better if I show you. By the end of this chapter you'll have the beginnings of an idea of what Combine is and how it works.

This is a journey that will change the way you write code.

I'm really loving working with Combine. As I wrote this book and began each chapter I thought, "I really love the ideas I'm going to cover in this chapter."

When you begin with Combine, you scratch your head and think, "I don't really see the point of this. It doesn't seem necessary. I already know ways to do all of this."

And that's true.

But once you use Combine you see so many places in your code that can be clarified by adopting it.

Enough preamble, let's begin.

In this chapter we begin with an example that is so tiny that you'll think, "this is stupid."

We'll build a simple app that displays a random number between 1 and 100 every time you tap a button.

A hero's journey needs to begin where the hero (that's you) is.

As I said, I assume you are a developer who is comfortable writing in Swift for one of Apple's platforms but I don't know whether you are using UIKit/Cocoa or SwiftUI.

I'll begin with UIKit because I want to stress that although Combine and SwiftUI work beautifully together, you can get started with Combine even if you're not using SwiftUI yet. If you don't use UIKit, don't worry, I'll explain just enough to get us started.

Anyway, back to our example.

Having a button that displays a random number is not the point of the example.

The point is to use that simple app to explain the four pillars of Combine: Publishers, Subscribers, Subscriptions, and Operators.

The first seven sections of this chapter will give you a gentle introduction to the fundamentals of Combine.

The final two sections are where I tuck the front-matter. That's where you'll find the road map to what's covered in this book, various links and credits, and version information. It's a personal quirk that I don't place those in the front of the book because I want us to get started right away with our adventure.

Later chapters will include theory and definitions and a ton of examples.

We'll have plenty of time in the rest of the book to look at individual trees. In this chapter, we begin our journey with a look at the forest.

Without Combine

In this section we look at a simple example that doesn't use Combine at all.

Sample Code

Head to https://github.com/editorscut/ec011CombineKickstart for the code that accompanies this book.

The code download has one folder for each chapter. This chapter's folder is *01*.

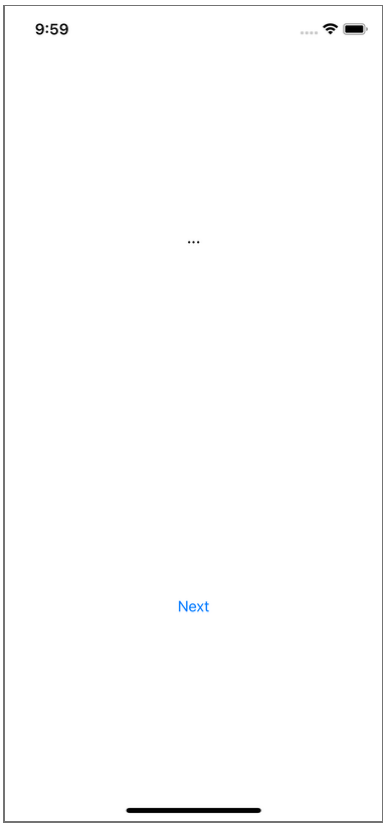
Inside *01* you'll find a folder for each section. In this chapter you'll also see a folder labeled *_StartHere*. Each of the other sections contains the finished code described in the corresponding section of the book.

_StartHere contains two projects. Open the one in the *Forest* folder. We'll look at *EnchantedForest* in the next section.

Run the app. You should check that you've chosen a simulator as it will default to your device if it is attached to your Mac. To run enter Command-R, select the menu item Product > Run, or tap the right triangle in the top left of Xcode as shown below.

In the figure above you can see that I've chosen the iPhone 12 mini simulator. Also, the deployment target for the code download is iOS 14.3. The code should run on earlier dot releases of iOS, but it has been tested on this release.

You should see a label containing three dots and below it a button with the title "Next".



Next, let's look at the relevant code.

The ViewController

I've implemented the UI using a storyboard that you can find inside of the *Support* group if you want to see it. There's nothing particularly interesting about it, but if you've never seen storyboards, this one contains a `UILabel` and a `UIButton` inside of a `UIStackView` that is used to position them.

The important thing is that storyboards include a mechanism that allows us to connect the button and label to our code so we can respond to button taps and send the label messages to update what it is displaying.

When the user taps the button we need to perform an action. We do this by connecting the button to a method that is designated an `IBAction`. When the button is tapped, this method is called.

Similarly, if we want to change the label's text, we connect the label to a property that is designated an `IBOutlet`. This property is our local handle to the actual label.

I've made the appropriate connections to the code that you'll find in the *ViewController.swift* file.

I've named the action `next()` and the outlet `label`.

01/01/Forest/Forest/ViewController.swift

```
import UIKit

class ViewController: UIViewController {
  @IBOutlet private weak var label: UILabel!

  @IBAction private func next(_ sender: UIButton) {
  }
}
```

Before we implement the action, note that I've listed the path as *01/01/Forest/...* as a caption above the code. That's because the completed version of the code we're building in this section will be found in the code for chapter 1, section 1: *01/01*. All code listings for code we will view or add to will be captioned to make it easier for you to locate the appropriate file.

The Action

Here's what we're going to do when the button is tapped:

- Choose a random `Int` between 1 and 100.
- Use `description` to get the `String` corresponding to that `Int`.
- Set `label`'s `text` property to equal this `String`.

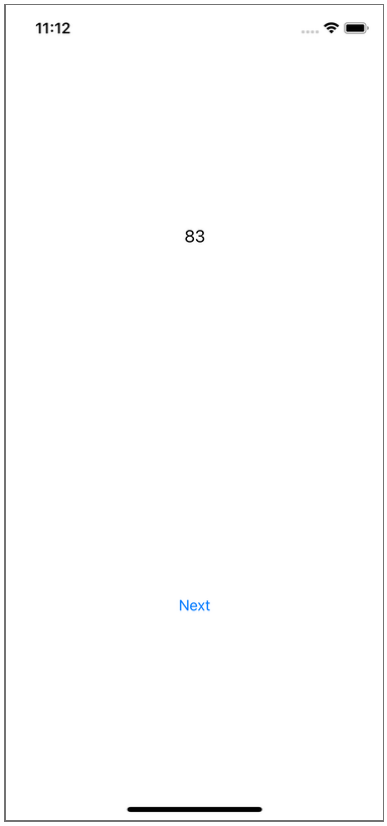
We'll accomplish all of that in a single line. Add the highlighted code below to the body of `next()`.

01/01/Forest/Forest/ViewController.swift

```
class ViewController: UIViewController {
  @IBOutlet private weak var label: UILabel!

  @IBAction private func next(_ sender: UIButton) {
    label.text = Int.random(in: 1...100).description
  }
}
```

Run the app again. Tap the button a few times.



Every time you tap the button a random number will be displayed by the label.

So...

Between now and the end of the chapter we're going to add more than a dozen lines of code to this example and several layers of indirection and it will never do more than it does right now with this single line of code.

Combine doesn't shine in this simple example.

You see the power of Combine in a real app where you are making network calls and communicating with a model of some sort and kicking off one or more actions from a change in state.

We'll get to all that in time.

For now, we're using this simple example to explore the fundamentals of Combine.

We'll come back to this example after we take a quick moment to implement this same app in SwiftUI. Again, if you haven't done SwiftUI yet, don't worry. We'll only look at enough SwiftUI to understand the example.