



First Edition

A SwiftUI Kickstart

DANIEL H STEINBERG



Introducing the
SwiftUI User Interface Framework

Editors Cut

A SwiftUI Kickstart

Introducing The SwiftUI
User Interface Framework

by Daniel H Steinberg

Editors Cut

Copyright

"A SwiftUI Kickstart", by Daniel H Steinberg

Copyright © 2019-2020 Dim Sum Thinking, Inc. All rights reserved.

ISBN-13: 9 978-1-944994-00-6

Book Version

This is version 0.6 for Swift 5.3, Xcode 12, and iOS 14 released October 10, 2020.

Recommended Settings

The ePub is best viewed in scrolling mode using the original fonts. On smaller devices I also choose landscape. If you view this book in Apple's Books app, choose "Let lines break naturally".

Legal

Every precaution was taken in the preparation of this book. The publisher and author assume no responsibility for errors and omissions, or for damages resulting from the use of the information contained herein and in the accompanying code downloads.

The sample code is intended to be used to illustrate points made in the text. It is not intended to be used in production code.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks or service marks. Where those designations appear in this book, and Dim Sum Thinking, Inc. was aware of the trademark claim, the designations have been printed with initial capital letters or in all capitals.

This book uses terms that are registered trademarks of Apple Inc. for which the terms of use don't permit rendering them in all caps or initial caps. You can view a complete list of the trademarks and registered trademarks of Apple Inc. at <http://www.apple.com/legal/trademark/appletmlist.html>.

The Editor's Cut name and logo are registered trademarks of Dim Sum Thinking, Inc.

Bindings

In this section we use `@Bindings` to modify a value that someone else owns. In this example we use a `Picker` to choose how much we increment `currentValue` by when we tap the right and left arrows.

Set up

Introduce an `Int` named `increment` that we'll set with a `Picker`. Because `increment` can change, it must be wrapped with `@State`. We initialize `increment` to 10.

03/06/Action/Action/ContentView.swift

```
struct ContentView {
    @State private var currentValue = 0
    @State private var increment = 10
}
```

Modify `back` and `forward` to use `increment`.

03/06/Action/Action/ContentView.swift

```
extension ContentView {  
    private func back() {  
        currentValue -= increment  
    }  
    private func forward() {  
        currentValue += increment  
    }  
}
```

Run the app. The arrows should increment `currentValue` by 10 for now.

Add a picker

A `Picker` is a SwiftUI component that is bound to an `Int`.

We'll dip into the details at the end of this chapter but for now you need to know that a property wrapper such as `@State` wraps a value. In the case of `increment` there is `increment` which is an `Int` and `$increment` which in this case is a `Binding<Int>`.

Here's how we use it in our picker.

03/06/Action/Action/ContentView.swift

```
extension ContentView: View {
    var body: some View {
        VStack {
            IntDisplay(value: currentValue)
            HStack {
                SymbolButton("arrow.left",
                    action: back)
                    .padding()
                SymbolButton("arrow.right",
                    action: forward)
                    .padding()
            }
            Picker("Choose the increment",
                selection: $increment){
                ForEach(0..<5) {index in
                    Text("by \(index)")
                }
            }
            .padding()
        }
    }
}
```

The selection parameter is what we use to connect the `Picker` to the `Int` it is bound to. Here we pass `$increment` which allows it to change the value of `increment` automatically.

The choices in a `Picker` are 0, 1, ... one less than the number of possible choices. In other words, just like the indices in an array. The `ForEach` returns a different `View` that will be presented in our `Picker` and for each `index` in that range we return a `Text` that contains "by" followed by the `index`.

The result looks like this.



Note that the row with "by 1" is selected because 1 is the default value for `increment`.

We need to make some changes.

Picker style

We can change the picker style from being a scroll type to a segmented button type like this.

03/06/Action/Action/ContentView.swift

```
extension ContentView: View {
    var body: some View {
        VStack {
            IntDisplay(value: currentValue)
            HStack {
                SymbolButton("arrow.left",
                            action: back)
                    .padding()
                SymbolButton("arrow.right",
                            action: forward)
                    .padding()
            }
            Picker("Choose the increment",
                  selection: $increment){
                ForEach(0..<5) {index in
                    Text("by \(index)")
                }
            }
            .pickerStyle(SegmentedPickerStyle())
            .padding()
        }
    }
}
```

With that simple change we've dramatically changed the look. Note that "by 1" is still the initial selection.



Let's make some more changes so that our increments aren't necessarily consecutive `Ints`.

Improving increment

I'd like the choices for the increment to be 1, 2, 3, 5, and 10 with 1 being the default value.

Introduce an array with these values and change out bound property to be their index.

03/06/Action/Action/ContentView.swift

```
let increments = [1, 2, 3, 5, 10]

struct ContentView {
    @State private var currentValue = 0
    @State private var incrementIndex = 0
}
```

Of course we need to adjust `back` and `forward`.

03/06/Action/Action/ContentView.swift

```
extension ContentView {
    private func back() {
        currentValue -= increments[incrementIndex]
    }
    private func forward() {
        currentValue += increments[incrementIndex]
    }
}
```

Finally, we need to adjust our `Picker`. It is now bound to `incrementIndex`, the range is defined to be the indices of the array,

and the displayed value for each button is the corresponding element of the array.

03/06/Action/Action/ContentView.swift

```
extension ContentView: View {
    var body: some View {
        VStack {
            IntDisplay(value: currentValue)
            HStack {
                SymbolButton("arrow.left",
                             action: back)
                    .padding()
                SymbolButton("arrow.right",
                             action: forward)
                    .padding()
            }
            Picker("Choose the increment",
                  selection: $incrementIndex){
                ForEach(increments.indices) {index in
                    Text("by \(increments[index])")
                }
            }
            .pickerStyle(SegmentedPickerStyle())
            .padding()
        }
    }
}
```

The labels for the picker and how it is used changes. Underneath the values for each selection is still 0..<5, we've just changed their effect on the app.



Run the app and use the picker to choose the value by which we increment and everything works as it should.

In the next section let's extract our [Picker](#) and explore another example of bindings.