From UIKit to SwiftUI

Veronica Ray

Senior Software Engineer at Compass

Goal

Rewrite a range slider from UIKit and Core Graphics into SwiftUI

Compass RangeSeekSlider

- ☞ Based on an existing open source project
- ☞ Heavily modified to remove features we didn't need and to follow our coding style

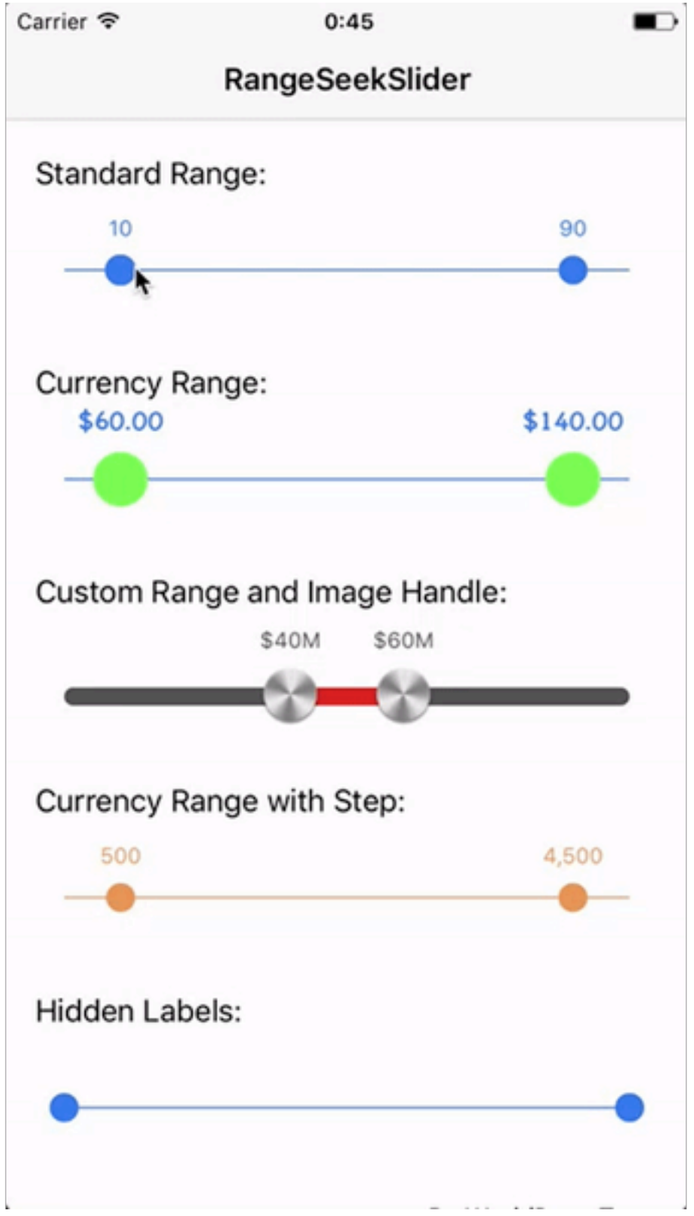
WorldDownTown / **RangeSeekSlider**

Watch 15 | Star 362 | Fork 169

Code | Issues 31 | Pull requests 23 | Projects 0 | Wiki | Security | Insights

RangeSeedSlider provides a customizable range slider like a UISlider.

swift uislider ios uikit xcode


Carrier
0:45
RangeSeekSlider
Standard Range:
10
90
Currency Range:
$60.00
$140.00
Custom Range and Image Handle:
$40M
$60M
Currency Range with Step:
500
4,500
Hidden Labels:

	Lines of code
RangeSeekSlider	360
RangeSeekSliderViewModel	120

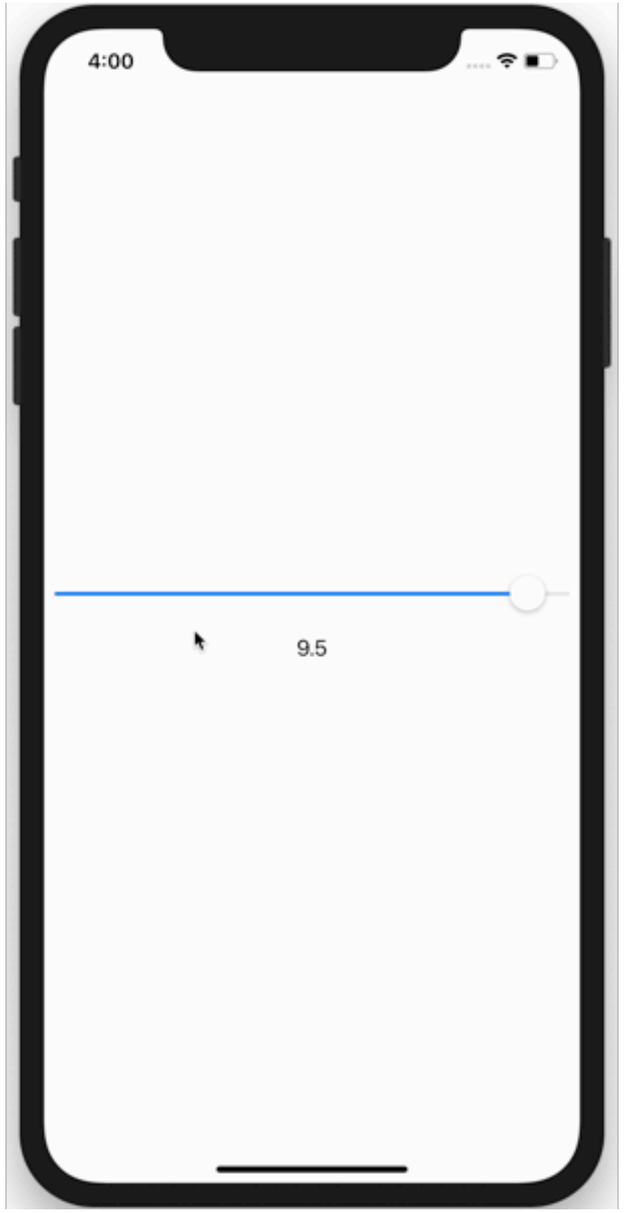
Shapes
Gestures

Start Simple And Get The Little Details Right

Native UISlider

Goals

- ☞ Correct padding on left and right side
- ☞ Formatted text that displays the selected value


4:00
9.5

Dependency Injection With Views

This Will Not Compile

```
struct TestSlider : View {
    @State var selectedValue: CGFloat = 0.0
    @State var numberFormatter: NumberFormatter

    init(numberFormatter: NumberFormatter = NumberFormatter.createNumberFormatter()) {
        self.numberFormatter = numberFormatter
    }

    var body: some View {
        // removed for brevity
    }
}
```

“`@State` variables in SwiftUI should not be initialized from data you pass down through the initializer...”

—Joe Groff, Senior Swift Compiler Engineer, Swift Forums

“...since the model is maintained outside of the view, there is no guarantee that the value will really be used.”

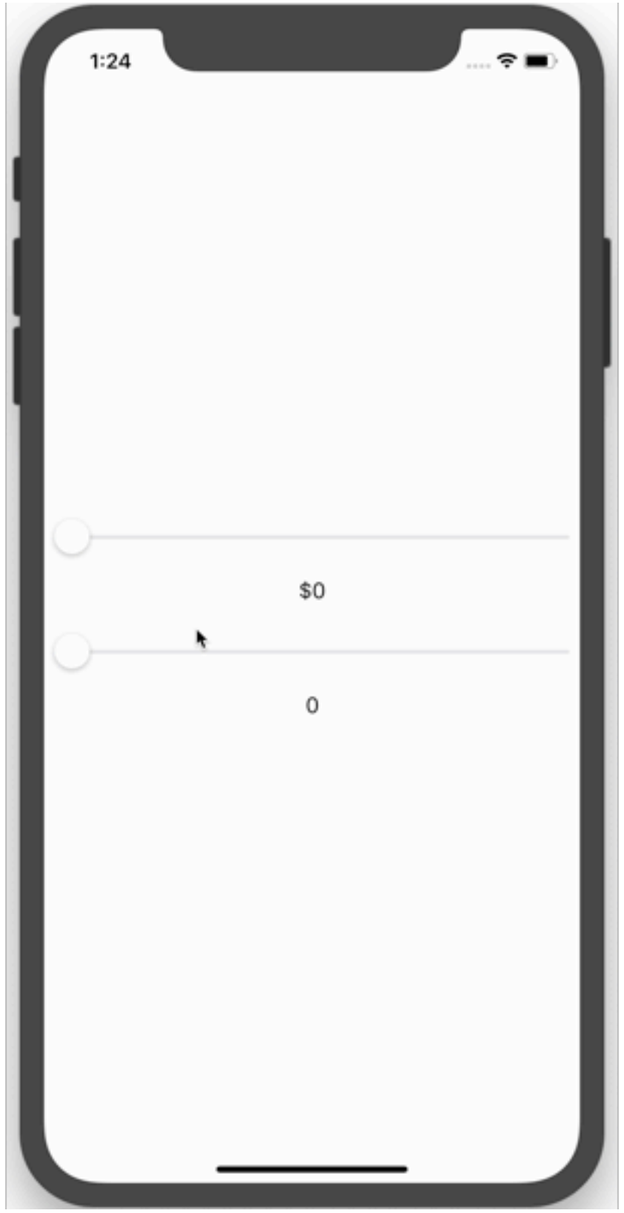
—Joe Groff, Senior Swift Compiler Engineer, Swift Forums

`@State` Is Primarily Intended For Small-Scale UI State

```swift
struct TestSlider: View {
    @State private var selectedValue: CGFloat = 0.0
    private let numberFormatter: NumberFormatter
    private let minValue: CGFloat
    private let maxValue: CGFloat
    private let step: CGFloat

    init(rangeType: RangeType) {
        self.numberFormatter = rangeType.numberFormatter
        self.minValue = rangeType.minValue
        self.maxValue = rangeType.maxValue
        self.step = rangeType.step
    }

    var body: some View {
        VStack {
            Slider(value: $selectedValue, from: minValue, through: maxValue, by: step)
                .padding()
            Text(numberFormatter.string(from: selectedValue as NSNumber) ?? "")
        }
    }
}
```


1:24
$0
0

Why Not Use `@BindableObject`?

- ☞ numberFormatter, min, max and step aren't going to change once you initialize the slider
- ☞ Only one view needs access to these values

CGFloats?!

I thought we were breaking from tradition...

And leaving behind old baggage that didn't serve us well...

Nick Lockwood
@nicklockwood

I'm a bit disappointed that SwiftUI is still using CGPoint, CGRect, etc. The ergonomics of using CGFloat with Swift are horrible, can't we just have new geometric primitives based on Doubles?

11:29 AM · Jul 9, 2019 · Twitter Web App

15 Retweets **157** Likes

Joe Groff's Response

- ☞ CGFloat is still single-precision in arm64_32, which is necessary for any retina display, or a non-retina display bigger than 1024×768.
- ☞ It's too late for shipping ABIs.
- ☞ "change CGFloat.h so that CGFloat is always double on not-yet-defined platforms" might be a good action to take.

Simple Gestures

```swift
override func beginTracking(_ touch: UITouch, with event: UIEvent?) -> Bool {
    super.beginTracking(touch, with: event)
    // calculations

    guard isTouchingLeftHandle || isTouchingRightHandle else {
        return false
    }
    // assign handleTracking to .left or .right

    return true
}
```

```swift
override func continueTracking(_ touch: UITouch, with event: UIEvent?) -> Bool {
    super.continueTracking(touch, with: event)
    guard handleTracking != .none else {
        return false
    }

    let location = touch.location(in: self)

    var percentage: CGFloat = 0
    var selectedValue: CGFloat = 0
    percentage = (location.x - sliderLine.frame.minX - handleDiameter / 2) / (sliderLine.frame.maxX - sliderLine.frame.minX)
    selectedValue = max(percentage * (viewModel.maxValue - viewModel.minValue) + viewModel.minValue, viewModel.minValue)

    switch handleTracking {
    case .left:
        viewModel.selectedMinValue = min(selectedValue, viewModel.selectedMaxValue)
    case .right:
        viewModel.selectedMaxValue = max(selectedValue, viewModel.selectedMinValue)
    case .none:
        break
    }

    refresh()

    return true
}
```

```
override func endTracking(_ touch: UITouch?, with event: UIEvent?) {
    super.endTracking(touch, with: event)
    handleTracking = .none
    initialTouchPoint = CGPoint.zero
    strokePhase = .notStarted
    trackedTouch = nil
}
```

“SwiftUI doesn't invoke the updating callback when the user ends or cancels a gesture. Instead, the gesture state property automatically resets its state back to its initial value.”

—SwiftUI Documentation

“SwiftUI only invokes the `onEnded(_:)` callback when the gesture succeeds.”

—***SwiftUI Documentation***

```swift
var body: some View {
        let minimumLongPressDuration = 0.5
        let longPressDrag = LongPressGesture(minimumDuration: minimumLongPressDuration)
            .sequenced(before: DragGesture())
            .updating($dragState) { value, state, transaction in
                switch value {
                // Long press begins.
                case .first(true):
                    state = .pressing
                // Long press confirmed, dragging may begin.
                case .second(true, let drag):
                    state = .dragging(translation: drag?.translation ?? .zero)
                // Dragging ended or the long press cancelled.
                default:
                    state = .inactive
                }
            }
            .onEnded { value in
                guard case .second(true, let drag?) = value else { return }
                self.viewState.width += drag.translation.width
                self.viewState.height += drag.translation.height
            }
```


ADVENTURE BOUND

First impressions of SwiftUI

June 8, 2019 by Matt Gallagher
Tags: app-design, cocoa

A little over a month ago, I released CwlViews and then followed up with an article suggesting that Apple might be about to release their own declarative views library. At WWDC this week, they did just that, releasing SwiftUI.

Moving From "How Do I Do X In Y?"

To "What's The New X?"

Where is `UIControl`?

Where are `beginTracking`/`continueTracking`/`endTracking`? I have to override them in my new class, right?

There's no more `UIControl`. Use a View.

We now call `updating`, `onChanged` and `onEnd` functions on `Gesture` instead of overriding any functions on our class.

RangeSeekSlider Without Gestures

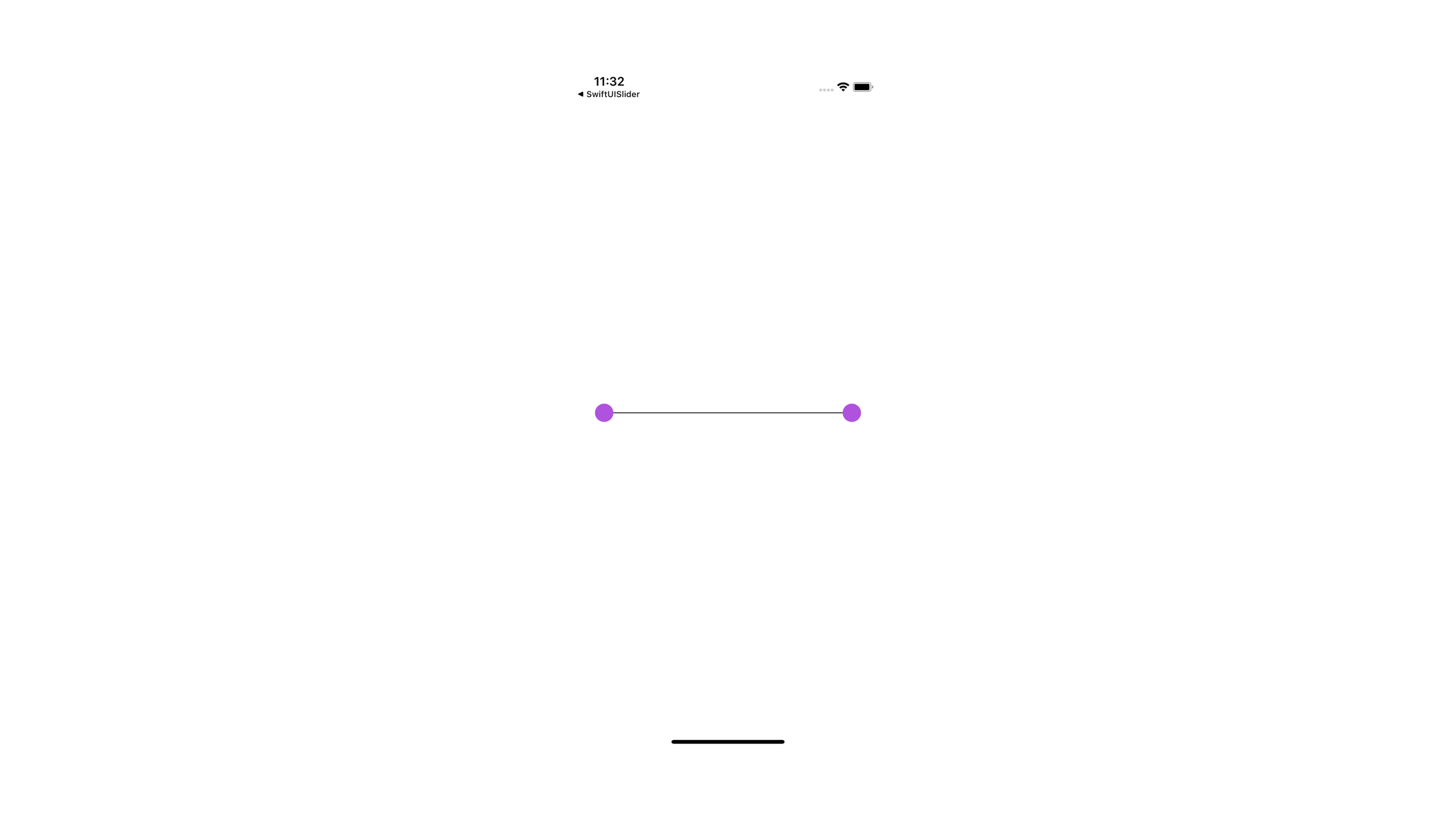
Building Custom Views in SwiftUI

Graphic effects and layout

Dave Abrahams
John Harper

- ☞ We no longer have a `UIKit` and `Core Graphics` separation
- ☞ Everything in SwiftUI is a `View`

```
struct ContentView : View {
    var body: some View {
        return HStack(spacing: 0) {
            Circle()
                .fill(Color.purple)
                .frame(width: 24, height: 24, alignment: .center)
                .zIndex(1)
            Rectangle()
                .frame(width: CGFloat(300.0), height: CGFloat(1.0), alignment: .center)
                .zIndex(0)
            Circle()
                .fill(Color.purple)
                .frame(width: 24, height: 24, alignment: .center)
                .zIndex(1)
        }
    }
}
```


11:32
◀ SwiftUISlider

RangeSeekSlider With Gestures

```swift
import SwiftUI

struct PriceContentView : View {
    @State private var selectedMinValue: CGFloat = RangeType.price.minValue
    @State private var selectedMaxValue: CGFloat = RangeType.price.maxValue
    @State private var leftHandleViewState = CGSize.zero
    @State private var rightHandleViewState = CGSize.zero
    private let numberFormatter = RangeType.price.numberFormatter
    private let minValue = RangeType.price.minValue
    private let maxValue = RangeType.price.maxValue
    private let step = RangeType.price.step
    private let lineWidth: CGFloat = 300.0
    private let handleDiameter: Length = 24
```

```
var body: some View {
    let leftHandleDragGesture = DragGesture(minimumDistance: 1, coordinateSpace: .local)
        .onChanged { value in
            guard value.location.x >= 0, value.location.x <= (self.lineWidth + self.handleDiameter) else {
                return
            }
            self.leftHandleViewState.width = value.location.x
            let percentage = self.leftHandleViewState.width/(self.lineWidth + self.handleDiameter)
            self.selectedMinValue = max(percentage * (self.maxValue - self.minValue) + self.minValue, self.minValue)
            self.selectedMinValue = CGFloat(roundf(Float(self.selectedMinValue / self.step))) * self.step
    }
```

```
let rightHandleDragGesture = DragGesture(minimumDistance: 1, coordinateSpace: .local)
    .onChanged { value in
        guard value.location.x <= 0, value.location.x >= -(self.lineWidth + self.handleDiameter) else {
            return
        }
        self.rightHandleViewState.width = value.location.x
        let percentage = 1 - abs(self.rightHandleViewState.width)/(self.lineWidth + self.handleDiameter)
        self.selectedMaxValue = max(percentage * (self.maxValue - self.minValue) + self.minValue, self.minValue)
        self.selectedMaxValue = CGFloat(roundf(Float(self.selectedMaxValue / self.step))) * self.step
}
```

```
        return
            VStack(spacing: 0) {
                HStack(spacing: 0) {
                    Circle()
                        .fill(Color.purple)
                        .frame(width: handleDiameter, height: handleDiameter, alignment: .center)
                        .offset(x: leftHandleViewState.width, y: 0)
                        .gesture(leftHandleDragGesture)
                        .zIndex(1)
                    Rectangle()
                        .frame(width: lineWidth, height: CGFloat(1.0), alignment: .center)
                        .zIndex(0)
                    Circle()
                        .fill(Color.purple)
                        .frame(width: handleDiameter, height: handleDiameter, alignment: .center)
                        .offset(x: rightHandleViewState.width, y: 0)
                        .gesture(rightHandleDragGesture)
                        .zIndex(1)
                }
                Text("Selected min value: \(numberFormatter.string(from: selectedMinValue as NSNumber) ?? "")")
                Text("Selected max value: \(numberFormatter.string(from: selectedMaxValue as NSNumber) ?? "")")
            }
    }
}
```


4:29
Selected min value: $0
Selected max value: $7,750,000

```
private func xPositionAlongLine(for value: CGFloat) -> CGFloat {
    let percentage = percentageAlongLine(for: value)
    let maxMinDif = sliderLine.frame.maxX - sliderLine.frame.minX
    let offset = percentage * maxMinDif
    return sliderLine.frame.minX + offset
}

private func percentageAlongLine(for value: CGFloat) -> CGFloat {
    guard viewModel.minValue < viewModel.maxValue else {
        return 0
    }
    let maxMinDif = viewModel.maxValue - viewModel.minValue
    let valueSubtracted = value - viewModel.minValue
    return valueSubtracted / maxMinDif
}
```

Lines Of Code	UIKit	SwiftUI
Determine which handle was being dragged	20	0
Set the x position of the handle during the drag gesture	14	6

Continuously Refine The UI

- ☞ Change `minimumDistance` of gestures to 1
- ☞ Guard against touch location to make sure handles are always on the line
- ☞ Formatted text that displays the selected min and max values

Your Knowledge Portfolio


Dow Jones
28000
26000
24000
22000
20000
Punkte
J J A S O N D J F M A M
vom Vortag, Veränderung in Prozent zum Vortag
2874,4 -0,07%
16.431,0 -0,08%
2882,3 -2,48%
9659,1 -0,02%
3140,1 -0,21%
2867,2 -0,51%
-0,19%
China

SwiftUI Should Be Open Sourced

“...the nature of larger declarative systems is such that API documentation will never fill-in all the details...”

—"First impressions of SwiftUI" by Matt Gallagher

“...there is too much behavior that does not manifest through the interface.”

—***"First impressions of SwiftUI" by Matt Gallagher***

www.nerdonica.com

@nerdonica