SwiftUI & Property Wrapper

Swift UI Basics and Property Wrapper Deep Dive

Ratnesh Jain

UIKit

Imperative Syntax

- Explicitly tell UIKit about looks and position of UI element step by step.
- Need to implement UI element for specific target platform.
  - iOS, iPadOS, watchOS, tvOS, macOS, visionOS …
- Need to layout manually (AutoLayout)

SwiftUI

Declarative Syntax

- We only need to declare
- UI is function of State
  - To change UI, must change State
  - If state is changed from external event, UI will be in sync 🪄
- SwiftUI supports Multi-platform development (platform)
  - SwiftUI will adjust the rendering of UI for target platform 😎
- No Autolayout 🥺

SwiftUI Basics

SwiftUI Basics

Agenda

- SwiftUI View Types
- Property Wrapper
- Property Wrapper in SwiftUI View
- SwiftUI’s Magic*
- Code Study of Property Wrapper with SwiftUI Auto update

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Ratnesh Jain

What are the “types” of Views in SwiftUI

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Text
Image
List
ScrollView
ForEach
Color
Shape
TapGesture
DragGesture
MagnifyGesture
RotateGesture
SpatialTapGesture
ExclusiveGesture
Menu
Label
ModifiedContent
MultiDatePicker
NavigationLink
NavigationSplitView
NavigationStack
NavigationView
OffsetShape
OutlineGroup
OutlineSubgroupChildren
PasteButton
PhaseAnimator
Picker
PlaceholderContentView
PlaceholderTextShapeStyle
PresentedWindowContent
ProgressView
RenameButton
ScrollViewReader
Section
SecureField
ShareLink
Slider
Spacer
Stepper
StrokeBorderShapeView
StrokeShapeView
SubscriptionView
TabView
Table
Text
TextEditor
TextField
TimelineView
Toggle
TupleView
UIViewRepresentable
VStack
ViewThatFits
WindowGroup
Optional
Never
AsyncImage
Button
EditButton

Ratnesh Jain

Text
Image
List
ScrollView
ForEach
Color
Shape
TapGesture
DragGesture
MagnifyGesture
RotateGesture
SpatialTapGesture
ExclusiveGesture
Menu
Label
ModifiedContent
MultiDatePicker
NavigationLink
NavigationSplitView
NavigationStack
NavigationView
OffsetShape
OutlineGroup
OutlineSubgroupChildren
PasteButton
PhaseAnimator
Picker
PlaceholderContentView
PlaceholderTextShapeStyle
PresentedWindowContent
ProgressView
RenameButton
ScrollViewReader
Section
SecureField
ShareLink
Slider
Spacer
Stepper
StrokeBorderShapeView
StrokeShapeView
SubscriptionView
TabView
Table
Text
TextEditor
TextField
TimelineView
Toggle
TupleView
UIViewRepresentable
VStack
ViewThatFits
WindowGroup
Optional
Never
AsyncImage
Button
EditButton

Ratnesh Jain

Not Easy to learn All at Once.

let's Categorise

Text
Image
List
ScrollView
ForEach
Color
Shape
TapGesture
DragGesture
MagnifyGesture
RotateGesture
SpatialTapGesture
ExclusiveGesture
Menu
Label
ModifiedContent
MultiDatePicker
NavigationLink
NavigationSplitView
NavigationStack
NavigationView
OffsetShape
OutlineGroup
RenameButton
ScrollViewReader
Section
SecureField
ShareLink
Slider
Spacer
Stepper
StrokeBorderShapeView
StrokeShapeView
SubscriptionView
TabView
Table
Text
TextEditor
TextField
TimelineView
Toggle
TupleView
UIViewRepresentable
VStack
ViewThatFits
WindowGroup
Optional
Never
AsyncImage
Button
EditButton

Ratnesh Jain

SwiftUI

View Types

- Primitive Views
- Container Views
- View Modifiers
- View Readers

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

View

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

View

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

View

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

ViewModifier

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

ViewModifier

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List {
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

View start with CAPITAL case

ViewModifier starts with .lowerCase

```swift
import SwiftUI

@MainActor
struct ContentView: View {
    @State var viewModel: PostsViewModel = .init()

    var body: some View {
        List
            ForEach(viewModel.posts, id: \.id) { post in
                Text(post.title)
            }
        }
        .task {
            await viewModel.fetchPosts()
        }
        .navigationTitle("Posts")
    }
}

#Preview {
    NavigationStack {
        ContentView()
    }
}
```

Almost all of code inside "body" is one kind of a View

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- **Color**
- Divider
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- **Divider**
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- **Divider**
- Spacer
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- **Spacer**
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- **Spacer**
- Gestures
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- **Gestures**
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- **Gestures**
- Material

SwiftUI

Primitive Views

- Text
- Image
- Color
- Divider
- Spacer
- Gestures
- **Material**

Never

A type has no values and can’t be constructed

```
@frozen enum Never {}
```

Never

A function will Never Return

```
// An infinite loop never returns.
func forever() -> Never {
    while true {
        print("I will print forever.")
    }
}

// Calling fatalError(_:file:line:) unconditionally stops the program.
func crashAndBurn() -> Never {
    fatalError("Something very, very bad happened")
}

let favoriteNumber: Result<Int, Never> = .success(42)
switch favoriteNumber {
case .success(let value):
    print("My favorite number is", value)
}
```

Never

A function will Never Return

```
// An infinite loop never returns.
func forever() -> Never {
    while true {
        print("I will print forever.")
    }
}

// Calling fatalError(_:file:line:) unconditionally stops the program.
func crashAndBurn() -> Never {
    fatalError("something very very bad happened")
}

let favoriteNumber: Result<Int, Never> = .success(42)
switch favoriteNumber {
case .success(let value):
    print("My favorite number is", value)
}
```

A function with Never return type = nonreturning function

Never

A function will Never Return

```
// An infinite loop never returns.
func forever() -> Never {
    while true {
        print("I will print forever.")
    }
}

// Calling fatalError() or fatal(_:file:line:) unconditionally stops the program.
func crashAndBurn() -> Never {
    fatalError("Something very, very bad happened")
}

let favoriteNumber: Result<Int, Never> = .success(42)
switch favoriteNumber {
case .success(let value):
    print("My favorite number is", value)
}
```

- Closures
- Methods
- Computed Properties
- Subscript

can be nonreturning

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Text : View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property`` property.
    public typealias Body = Never
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Image : View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property`` property.
    public typealias Body = Never
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Color : View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property`` property.
    public typealias Body = Never
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
@frozen public struct Spacer : View {

    /// The minimum length this spacer can be shrunk to, along the axis or axes
    /// of expansion.
    ///
    /// If `nil`, the system default spacing between views is used.
    public var minLength: CGFloat?
    @inlinable public init(minLength: CGFloat? = nil)

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property`` property.
    public typealias Body = Never
}
```

SwiftUI

Primitive Views

```swift
@available(iOS 13.0, macOS 10.15, tvOS 16.0, watchOS 6.0, *)
public struct TapGesture : Gesture {

    /// The required number of tap events.
    public var count: Int

    /// Creates a tap gesture with the number of required taps.
    ///
    /// - Parameter count: The required number of taps to complete the tap
    ///   gesture.
    public init(count: Int = 1)

    /// The type of gesture representing the body of `Self`.
    public typealias Body = Never

    /// The type representing the gesture's value.
    public typealias Value = Void
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 16.0, watchOS 6.0, *)
public struct TapGesture : Gesture {

    /// The required number of taps events.
    public var count: Int

    /// Creates a tap gesture with the number of required taps.
    ///
    /// - Parameter count: The required number of taps to complete the tap
    ///    gesture.
    public init(count: Int = 1)

    /// The type of gesture representing the body of `Self`.
    public typealias Body = Never

    /// The type representing the gesture's value.
    public typealias Value = Void
}
```

Why these all these Views' "Body" type declared as Never

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public protocol View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property`` property.
    associatedtype Body : View

    @ViewBuilder @MainActor var body: Self.Body { get }
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public protocol View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property`` property.
    associatedtype Body : View

    @ViewBuilder @MainActor var body: Self.Body { get }
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public protocol View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property`` property.
    associatedtype Body : View

    @ViewBuilder @MainActor var body: Self.Body { get }
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public protocol View {

    /// The type of view representing the body of this view.
    ///
    /// When you create a custom view, Swift infers this type from your
    /// implementation of the required ``View/body-swift.property`` property.
    associatedtype Body : View

    @ViewBuilder @MainActor var body: Self.Body { get }
}
```

🔄 Recursion in Protocol Conformance

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never {

    /// The type for the internal content of this `AccessibilityRotorContent`.
    public typealias Body = Never

    /// The internal content of this `AccessibilityRotorContent`.
    public var body: Never { get }
}


@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never : View {
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never {

    /// The type for the internal content of this `AccessibilityRotorContent`.
    public typealias Body = Never

    /// The internal content of this `AccessibilityRotorContent`.
    public var body: Never { get }
}


@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never : View {
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never {

    /// The type for the internal content of this `AccessibilityRotorContent`.
    public typealias Body = Never

    /// The internal content of this `AccessibilityRotorContent`.
    public var body: Never { get }
}


@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never : View {
}
```

SwiftUI

Primitive Views

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never {

    /// The type for the internal content of this `AccessibilityRotorContent`.
    public typealias Body = Never

    /// The internal content of this `AccessibilityRotorContent`.
    public var body: Never { get }
}


@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never : View {
}
```

SwiftUI

Primitive Views

```
@available(iOS 3.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never {

    /// The type for the internal content of this `AccessibilityRotorContent`.
    public typealias Body = Never

    /// The internal content of this `AccessibilityRotorContent`.
    public var body: Never { get }
}

@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Never : View {
}
```

Whenever SwiftUI sees Never, it uses other Core technologies for onscreen rendering.

Core Animation, Core Graphics, Core Text, UIKit*, AppKit*, Metal …

SwiftUI

Uses Other Technology for Rendering Example

```
@available(iOS 13.0, tvOS 13.0, *)
@available(macOS, unavailable)
@available(watchOS, unavailable)

public protocol UIViewRepresentable : View where Self.Body == Never {

    /// The type of view to present.
    associatedtype UIViewType : UIView

    @MainActor func makeUIView(context: Self.Context) -> Self.UIViewType

    @MainActor func updateUIView(_ uiView: Self.UIViewType, context: Self.Context)

    ...
}
```

SwiftUI

Uses Other Technology for Rendering Example

```
@available(iOS 13.0, tvOS 13.0, *)
@available(macOS, unavailable)
@available(watchOS, unavailable)

public protocol UIViewRepresentable : View where Self.Body == Never {

    /// The type of view to present.
    associatedtype UIViewType : UIView

    @MainActor func makeUIView(context: Self.Context) -> Self.UIViewType

    @MainActor func updateUIView(_ uiView: Self.UIViewType, context: Self.Context)

    ...
}
```

SwiftUI

Uses Other Technology for Rendering Example

```
@available(iOS 13.0, tvOS 13.0, *)
@available(macOS, unavailable)
@available(watchOS, unavailable)

public protocol UIViewRepresentable : View where Self.Body == Never {

    /// The type of view to present.
    associatedtype UIViewType : UIView

    @MainActor func makeUIView(context: Self.Context) -> Self.UIViewType

    @MainActor func updateUIView(_ uiView: Self.UIViewType, context: Self.Context)

    ...
}
```

SwiftUI

Uses Other Technology for Rendering Example

```
@available(iOS 13.0, tvOS 13.0, *)
@available(macOS, unavailable)
@available(watchOS, unavailable)

public protocol UIViewRepresentable : View where Self.Body == Never {

    /// The type of view to present.
    associatedtype UIViewType : UIView

    @MainActor func makeUIView(context: Self.Context) -> Self.UIViewType

    @MainActor func updateUIView(_ uiView: Self.UIViewType, context: Self.Context)

    ...
}
```

SwiftUI

Uses Other Technology for Rendering Example

```
@available(iOS 13.0, tvOS 13.0, *)
@available(macOS, unavailable)
@available(watchOS, unavailable)

public protocol UIViewRepresentable : View where Self.Body == Never {

    /// The type of view to present.
    associatedtype UIViewType : UIView

    @MainActor func makeUIView(context: Self.Context) -> Self.UIViewType

    @MainActor func updateUIView(_ uiView: Self.UIViewType, context: Self.Context)

    ...
}
```

SwiftUI

Uses Other Technology for Rendering Example

```
@available(iOS 13.0, tvOS 13.0, *)
@available(macOS, unavailable)
@available(watchOS, unavailable)

public protocol UIViewRepresentable : View where Self.Body == Never {

    /// The type of view to present.
    associatedtype UIViewType : UIView

    @MainActor func makeUIView(context: Self.Context) -> Self.UIViewType

    @MainActor func updateUIView(_ uiView: Self.UIViewType, context: Self.Context)

    ...
}
```

SwiftUI

Uses Other Technology for Rendering Example

```swift
struct AirPlayView: UIViewRepresentable {
    func makeUIView(context: Context) -> AVRoutePickerView {
        let view = AVRoutePickerView()
        return view
    }

    func updateUIView(_ uiView: AVRoutePickerView, context: Context) {}
}
```

SwiftUI

Uses Other Technology for Rendering Example

```
struct AirPlayView: UIViewRepresentable {
    func makeUIView(context: Context) -> AVRoutePickerView {
        let view = AVRoutePickerView()
        return view
    }

    func updateUIView(_ uiView: AVRoutePickerView, context: Context) {}
}
```

SwiftUI

Uses Other Technology for Rendering Example

```
struct AirPlayView: UIViewRepresentable {
    func makeUIView(context: Context) -> AVRoutePickerView {
        let view = AVRoutePickerView()
        return view
    }

    func updateUIView(_ uiView: AVRoutePickerView, context: Context) {}
}
```

SwiftUI

Uses Other Technology for Rendering Example

```swift
struct AirPlayView: UIViewRepresentable {
    func makeUIView(context: Context) -> AVRoutePickerView {
        let view = AVRoutePickerView()
        return view
    }

    func updateUIView(_ uiView: AVRoutePickerView, context: Context) {}
}
```

SwiftUI

Views

- Primitive Views
- Container Views
- View Modifiers
- View Readers

SwiftUI

Container Views

HStack	Grid	List	TabView	Label
VStack	LazyHGrid	ScrollView	NavigationView	Button
ZStack	LazyVGrid	ForEach	NavigationStack	Menu
LazyHStack			NavigationSplitView	
LazyVStack				

SwiftUI

Container Views

- HStack
- VStack
- ZStack
- LazyHStack
- LazyVStack
- Grid
- LazyHGrid
- LazyVGrid
- List
- ScrollView
- ForEach
- TabView
- NavigationView
- NavigationStack
- NavigationSplitView

- Label
- Button
- Menu

SwiftUI

Container Views

- HStack
- VStack
- ZStack
- LazyHStack
- LazyVStack
- Grid
- LazyHGrid
- LazyVGrid
- List
- ScrollView
- ForEach
- TabView
- NavigationView
- NavigationStack
- NavigationSplitView

- Label
- Button
- Menu

SwiftUI

Container Views

- HStack
- VStack
- ZStack
- LazyHStack
- LazyVStack
- Grid
- LazyHGrid
- LazyVGrid
- List
- ScrollView
- ForEach
- TabView
- NavigationView
- NavigationStack
- NavigationSplitView

- Label
- Button
- Menu

SwiftUI

Container Views

- HStack
- VStack
- ZStack
- LazyHStack
- LazyVStack
- Grid
- LazyHGrid
- LazyVGrid
- List
- ScrollView
- ForEach
- TabView
- NavigationView
- NavigationStack
- NavigationSplitView

- Label
- Button
- Menu

SwiftUI

Container Views

- HStack
- VStack
- ZStack
- LazyHStack
- LazyVStack
- Grid
- LazyHGrid
- LazyVGrid
- List
- ScrollView
- ForEach
- TabView
- NavigationView
- NavigationStack
- NavigationSplitView

- Label
- Button
- Menu

SwiftUI

Container Views

```
HStack {
    PlayerImageView(url: store.session?.coverURL)
        .id(store.session?.coverURL)
        .frame(width: 48, height: 48)

    PlaybackInfoView()

    Spacer()
    Button {
        store.send(.user(.playbackControlButtonTapped))
    } label: {
        store.status.controlView
            .symbolVariant(.fill)
            .imageScale(.large)
            .font(.largeTitle)
            .contentTransition(.interpolate)
    }
    .disabled(store.session == nil)
}
```

SwiftUI

Container Views

- HStack
- Grid
- List
- TabView
- VStack
- LazyHGrid
- ScrollView
- NavigationView
- ZStack
- LazyVGrid
- ForEach
- NavigationStack
- LazyHStack
- NavigationSplitView
- LazyVStack

- Label
- Button
- Menu

SwiftUI

Container Views

VStack

```
VStack(alignment: .leading, spacing: 0) {
    if let session = store.session {
        Text(session.displayTitle)
            .font(.callout)
            .bold()
        TimeElapsedLabel()
    } else {
        Text("Not Playing")
            .font(.callout)
            .bold()
    }
}
```

SwiftUI

Container Views

- HStack
- Grid
- List
- TabView
- VStack
- LazyHGrid
- ScrollView
- NavigationView
- ZStack
- LazyVGrid
- ForEach
- NavigationStack
- LazyHStack
- NavigationSplitView
- LazyVStack

- Label
- Button
- Menu

SwiftUI

Container Views

- HStack
- VStack
- ZStack
- LazyHStack
- LazyVStack
- Grid
- LazyHGrid
- LazyVGrid
- List
- ScrollView
- ForEach
- TabView
- NavigationView
- NavigationStack
- NavigationSplitView

- Label
- Button
- Menu

SwiftUI

Container Views

ZStack

```
ZStack {
    LibraryImageView(url: store.library?.overURL)
        .aspectRatio(1, contentMode: .fill)
        .frame(height: 100)

    PlayButton {
        store.send(.play(store.library?.episodeId))
    }
    .shadow(color: Color.secondary.opacity(0.6), radius: 8)
}
```

SwiftUI

Container Views

- HStack
- Grid
- List
- TabView
- VStack
- LazyHGrid
- ScrollView
- NavigationView
- ZStack
- LazyVGrid
- ForEach
- NavigationStack
- **LazyHStack**
- NavigationSplitView
- LazyVStack

- Label
- Button
- Menu

SwiftUI

Container Views

LazyHStack

```
LazyHStack {
    ForEach(items, id: \.stableId) { item in
        VStack(alignment: .leading, spacing: 8) {
            LibraryItemImage(library: item)
            LibraryInfoView(library: item)
        }
    }
}
```

SwiftUI

Container Views

- HStack
- Grid
- List
- TabView
- VStack
- LazyHGrid
- ScrollView
- NavigationView
- ZStack
- LazyVGrid
- ForEach
- NavigationStack
- LazyHStack
- NavigationSplitView
- LazyVStack

- Label
- Button
- Menu

SwiftUI

Container Views

- HStack
- Grid
- List
- TabView
- VStack
- LazyHGrid
- ScrollView
- NavigationView
- ZStack
- LazyVGrid
- ForEach
- NavigationStack
- LazyHStack
- NavigationSplitView
- LazyVStack

- Label
- Button
- Menu

SwiftUI

Container Views

ScrollView

```
ScrollView {
    ForEach(shelves) { shelf in
        Section {
            ShelfSection(shelf: shelf, store: store)
        } header: {
            Text(shelf.label)
                .bold()
                .frame(maxWidth: .infinity, alignment: .leading)
                .padding(.horizontal)
        }
    }
}
```

SwiftUI

Container Views

- HStack
- Grid
- List
- TabView
- VStack
- LazyHGrid
- ScrollView
- NavigationView
- ZStack
- LazyVGrid
- **ForEach**
- NavigationStack
- LazyHStack
- NavigationSplitView
- LazyVStack

- Label
- Button
- Menu

SwiftUI

Container Views

ForEach

```
ForEach(items, id: \.stableId) { item in
    VStack(alignment: .leading, spacing: 8) {
        LibraryItemImage(library: item)
        LibraryInfoView(library: item)
    }
    .frame(width: size)
}
```

SwiftUI

Container Views

- HStack
- Grid
- List
- TabView
- VStack
- LazyHGrid
- ScrollView
- NavigationView
- ZStack
- LazyVGrid
- ForEach
- NavigationStack
- LazyHStack
- NavigationSplitView
- LazyVStack

- Label
- Button
- Menu

SwiftUI

Container Views

TabView

```
TabView(selection: $store.selectedTab) {
    ForEach(store.tabs)) { tabStore in
        TabsView(store: tabStore)
            .tabItem {
                Label(
                    tabStore.label.title,
                    systemImage: tabStore.label.systemImage
                )
            }
    }
}
```

SwiftUI

Container Views

- HStack
- VStack
- ZStack
- LazyHStack
- LazyVStack
- Grid
- LazyHGrid
- LazyVGrid
- List
- ScrollView
- ForEach
- TabView
- NavigationView
- NavigationStack
- NavigationSplitView

- Label
- Button
- Menu

SwiftUI

Container Views

NavigationStack

```
NavigationStack(path: $store.path)) {
    HomeView(store: store.root))
        .navigationTitle("Home")
} destination: { destinationStore in
    PathDestinationView(store: destinationStore)
}
```

SwiftUI

Container Views

- HStack
- Grid
- List
- TabView
- VStack
- LazyHGrid
- ScrollView
- NavigationView
- ZStack
- LazyVGrid
- ForEach
- NavigationStack
- LazyHStack
- NavigationSplitView
- LazyVStack

- Label
- Button
- Menu

SwiftUI

Container Views

Label

```
Label("Apple", systemImage: "square.stack.3d.up.fill")

— OR —

Label {
    Text("Apple")
} icon: {
    Image(systemImage: “square.stack.3d.up.fill”)
}
```

SwiftUI

Container Views

HStack
Grid
List
TabView
VStack
LazyHGrid
ScrollView
NavigationView
ZStack
LazyVGrid
ForEach
NavigationStack
LazyHStack
NavigationSplitView
LazyVStack

Label
Button
Menu

SwiftUI

Container Views

Button

```
Button("Play") {
    store.send(.playButtonTapped)
}

—OR —

Button {
    store.send(.playButtonTapped)
} label: {
    Image(systemName: "play.fill")
}
```

SwiftUI

Container Views

HStack
Grid
List
TabView
VStack
LazyHGrid
ScrollView
NavigationView
ZStack
LazyVGrid
ForEach
NavigationStack
LazyHStack
NavigationSplitView
LazyVStack

Label
Button
Menu

SwiftUI

Container Views

Menu

```
Menu {
    Button {
        store.send(.librarySelected(id: "Apple"))
    } label: {
        Label("Apple", systemImage: "square.stack.3d.up.fill")
    }
    Button {
        store.send(.librarySelected(id: "google"))
    } label: {
        Label("Google", systemImage: "mail.stack")
    }
}
```

SwiftUI

Container Views

Menu

```
Menu {
    Button {
        store.send(.librarySelected(id: "Apple"))
    } label: {
        Label("Apple", systemImage: "square.stack.3d.up.fill")
    }
    Button {
        store.send(.librarySelected(id: "google"))
    } label: {
        Label("Google", systemImage: "mail.stack")
    }
}
```

SwiftUI

Views

- Primitive Views
- Container Views
- View Modifiers
- View Readers

SwiftUI

View Modifiers

A modifier that you apply to a view or another view modifier, producing a different version of the original value.

SwiftUI Documentation

https://developer.apple.com/documentation/swiftui/viewmodifier

SwiftUI

View Modifiers

Decoration

```
VStack {
    Image(systemName: "globe")
        .imageScale(.large)
        .foregroundStyle(.tint)
    Text("Hello, world!")
}
```

SwiftUI

View Modifiers

Decoration

```
VStack {
    Image(systemName: "globe")
        .imageScale(.large)
        .foregroundStyle(.tint)
    Text("Hello, world!")
}
.padding()
```

SwiftUI

View Modifiers

Decoration

```
VStack {
    Image(systemName: "globe")
        .imageScale(.large)
        .foregroundStyle(.tint)
    Text("Hello, world!")
}
.padding()
.background(Color.yellow)
```

SwiftUI

View Modifiers

Decoration

```
VStack {
    Image(systemName: "globe")
        .imageScale(.large)
        .foregroundStyle(.tint)
    Text("Hello, world!")
}
.padding()
.background(Color.yellow)
.clipShape(.rect(cornerRadius: 12, style: .continuous))
```

SwiftUI

View Modifiers

Decoration

Passing values via Environment / Preferences

```
struct ContentView: View {
    @State private var selection: Set<Int> = []
    var body: some View {
        List(selection: $selection) {
            ForEach(1...10, id: \.self) { index in
                Text("\(index)")
            }
        }
    }
}
```

SwiftUI

View Modifiers

Decoration

Passing values via Environment / Preferences

```
struct ContentView: View {
    @State private var selection: Set<Int> = []
    var body: some View {
        List(selection: $selection) {
            ForEach(1...10, id: \.self) { index in
                Text("\(index)")
            }
        }
        .environment(\.editMode, .constant(.active))
    }
}
```

SwiftUI

View Modifiers

Decoration

Passing values

Reading dimensions

```
.background {
    GeometryReader { proxy in
        Color.clear
            .onChange(of: proxy.size, initial: true) { oldValue, newV
                self.size = newValue
            }
            .onChange(of: proxy.frame(in: .global), initial: true) {
                self.position = newValue.origin
            }
    }
}
```

```swift
struct ContentView: View {
    @State private var position: CGPoint = .zero
    @State private var size: CGSize = .zero

    var body: some View {
        List {
            TopArt()
                .background {
                    GeometryReader { proxy in
                        Color.clear
                            .onChange(of: proxy.size, initial: true) { oldValue, newValue in
                                self.size = newValue
                            }
                            .onChange(of: proxy.frame(in: .global), initial: true) { oldValue, newValue in
                                self.position = newValue.origin
                            }
                    }
                }
            Text("Position: \(position.debugDescription)")
            Text("Size: \(size.debugDescription)")
        }
    }
}
```

SwiftUI

View Modifiers

Decoration

Passing values

Reading dimensions

Observing life cycles

```
List {
    ForEach(1...10, id: \.self) { index in
        Text("Item \(index)")
    }
}
.onAppear {
    print("On Appear")
}
.onDisappear {
    print("On Disappear Called")
}
.task {
    print("Creates new task on OnAppear and cancels it on Disappear")
}
```

SwiftUI

View Modifiers

Decoration

Passing values

Reading dimensions

Observing life cycles

View start with CAPITAL case

ViewModifier starts with .lowerCase

SwiftUI

Views

- Primitive Views
- Container Views
- View Modifiers
- View Readers

SwiftUI

View Readers

GeometryReader

ScrollViewReader

```
.background {
    GeometryReader { proxy in
        Color.clear
            .onChange(of: proxy.size, initial: true) { oldValue, newValue in
                self.size = newValue
            }
            .onChange(of: proxy.frame(in: .global), initial: true) { oldValue, newValue in
                self.position = newValue.origin
            }
    }
}
```

SwiftUI

View Readers

GeometryReader

ScrollViewReader

```
struct ContentView: View {
    var body: some View {
        ScrollViewReader(content: { proxy in
            List {
                ForEach(1...50, id: \.self) { index in
                    Text("Index: \(index)")
                }
            }
            .safeAreaInset(edge: .bottom) {
                Button("Scroll to 40") {
                    withAnimation(.smooth) {
                        proxy.scrollTo(40)
                    }
                }
                .buttonStyle(.borderedProminent)
            }
        })
    }
}
```

Ratnesh Jain

SwiftUI

View Readers

GeometryReader

ScrollViewReader

```
struct ContentView: View {
    var body: some View {
        ScrollViewReader(content: { proxy in
            List {
                ForEach(1...50, id: \.self) { index in
                    Text("Index: \(index)")
                }
            }
            .safeAreaInset(edge: .bottom) {
                Button("Scroll to 40") {
                    withAnimation(.smooth) {
                        proxy.scrollTo(40)
                    }
                }
                .buttonStyle(.borderedProminent)
            }
        })
    }
}
```

Ratnesh Jain

SwiftUI

View Readers

GeometryReader

ScrollViewReader

```
struct ContentView: View {
    var body: some View {
        ScrollViewReader(content: { proxy in
            List {
                ForEach(1...50, id: \.self) { index in
                    Text("Index: \(index)")
                }
            }
            .safeAreaInset(edge: .bottom) {
                Button("Scroll to 40") {
                    withAnimation(.smooth) {
                        proxy.scrollTo(40)
                    }
                }
                .buttonStyle(.borderedProminent)
            }
        })
    }
}
```

Ratnesh Jain

SwiftUI

View Readers

GeometryReader

ScrollViewReader

```
struct ContentView: View {
    var body: some View {
        ScrollViewReader(content: { proxy in
            List {
                ForEach(1...50, id: \.self) { index in
                    Text("Index: \(index)")
                }
            }
            .safeAreaInset(edge: .bottom) {
                Button("Scroll to 40") {
                    withAnimation(.smooth) {
                        proxy.scrollTo(40)
                    }
                }
                .buttonStyle(.borderedProminent)
            }
        })
    }
}
```

Ratnesh Jain

SwiftUI

View Readers

GeometryReader

ScrollViewReader

```
struct ContentView: View {
    var body: some View {
        ScrollViewReader(content: { proxy in
            List {
                ForEach(1...50, id: \.self) { index in
                    Text("Index: \(index)")
                }
            }
            .safeAreaInset(edge: .bottom) {
                Button("Scroll to 40") {
                    withAnimation(.smooth) {
                        proxy.scrollTo(40, anchor: .top)
                    }
                }
                .buttonStyle(.borderedProminent)
            }
        })
    }
}
```

Ratnesh Jain

SwiftUI

Views

- Primitive Views
- Container Views
- View Modifiers
- View Readers

Property Wrapper

Property Wrapper

Syntax

```
@propertyWrapper
struct InMemoryPropertyWrapper {

}
```

Property wrapper type 'InMemoryPropertyWrapper' does not contain a non-static property named 'wrappedValue'

Property Wrapper

Syntax

```
@propertyWrapper
struct InMemoryPropertyWrapper<T> {
    var wrappedValue: T
}
```

Property wrapper type 'InMemoryPropertyWrapper' does not contain a non-static property named 'wrappedValue'

Property Wrapper

Syntax

```
@propertyWrapper
struct InMemoryPropertyWrapper<Value> {
    var wrappedValue: Value
}
```

Property Wrapper

Syntax

```
@propertyWrapper
struct InMemoryPropertyWrapper<Value> {
    var wrappedValue: Value
}
```

But Why?

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int = 10
    var pageOffset: Int = 0

    init() {}

    func fetchPosts() async {
        ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int = 10
    var pageOffset: Int = 0

    init() {}

    func fetchPosts() async {
        ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
     ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
      ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            return (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            return (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
     ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            return (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            return (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
      ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
     ...
    }
}
```

Why Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageLimit")
        }
    }
    var pageOffset: Int {
        get {
            (UserDefaults.standard.value(forKey: "pageOffset") as? Int) ?? 0
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: "pageOffset")
        }
    }

    func fetchPosts() async {
     ...
    }
}
```

Why Property Wrapper

```
struct UserDefaultStore {

    var propertyValue: Int {
        get {
            (UserDefaults.standard.value(forKey: “pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: “pageLimit")
        }
    }
}
```

Why Property Wrapper

```
@propertyWrapper struct UserDefaultStore {

    var propertyValue: Int {
        get {
            (UserDefaults.standard.value(forKey: “pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: “pageLimit")
        }
    }
}
```

Why Property Wrapper

```
@propertyWrapper struct UserDefaultStore {

    var wrappedValue: Int {
        get {
            (UserDefaults.standard.value(forKey: “pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: “pageLimit")
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore {

    var wrappedValue: Int {
        get {
            (UserDefaults.standard.value(forKey: “pageLimit") as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: “pageLimit")
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore {
    let key: String

    var wrappedValue: Int {
        get {
            (UserDefaults.standard.value(forKey: key) as? Int) ?? 10
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore {
    let defaultValue: Int
    let key: String

    var wrappedValue: Int {
        get {
            (UserDefaults.standard.value(forKey: key) as? Int) ?? defaultValue
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let defaultValue: T
    let key: String

    var wrappedValue: T {
        get {
            (UserDefaults.standard.value(forKey: key) as? T) ?? defaultValue
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    var pageLimit: Int = 10
    var pageOffset: Int = 0

    init() {}

    func fetchPosts() async {
        ...
    }
}
```

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit", defaultValue: 10) var pageLimit: Int

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset", defaultValue: 0) var pageOffset: Int

    func fetchPosts() async {
        ...
    }
}
```

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset", defaultValue: 0) var p

    func fetchPosts() async {
        ...
    }
}
```

Extra argument 'wrappedValue' in call

Missing argument for parameter 'defaultValue' in property wrapper initializer; add 'wrappedValue' and 'defaultValue' arguments in '@UserDefaultStore(...)'

```
...
@AppStorage("item") var item: String = ""
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let defaultValue: T
    let key: String

    var wrappedValue: T {
        get {
            (UserDefaults.standard.value(forKey: key) as? T) ?? defaultValue
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

    init(key: String, defaultValue: T) {
        self.key = key
        self.defaultValue = defaultValue
    }

    var wrappedValue: T {
        get {
            UserDefaults.standard.value(forKey: key) as? T ?? defaultValue
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

How to Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

    init(key: String, defaultValue: T) {
        self.key = key
        self.defaultValue = defaultValue
    }

    init(wrappedValue: T, key: String) {
        self.defaultValue = wrappedValue
        self.key = key
    }

    var wrappedValue: T {
        get {
            UserDefaults.standard.value(forKey: key) as? T ?? defaultValue
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset", defaultValue: 0) var p

    func fetchPosts() async {
        ...
    }
}
```

Extra argument 'wrappedValue' in call

Missing argument for parameter 'defaultValue' in property wrapper initializer; add 'wrappedValue' and 'defaultValue' arguments in '@UserDefaultStore(...)'

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        ...
    }
}
```

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []


    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        ...
    }
}
```

Property wrapper cannot be applied to a computed property

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []


    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        ...
    }
}
```

Macro does not play well with property wrappers yet

How to Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        ...
    }
}
```

Property Wrapper

```
@MainActor @Observable class PostViewModel {
    var posts: [Post] = []

    @ObservationIgnored
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    @ObservationIgnored
    @UserDefaultStore(key: "pageOffset") var pageOffset: Int = 0

    func fetchPosts() async {
        ...
    }
}
```

Property Wrapper

```
struct PostView: View {
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    var body: some View {
        LabeledContent("PageLimit") {
            Text("\(pageLimit)")
        }
    }
}
```

Property Wrapper

```
struct PostView: View {
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    var body: some View {
        LabeledContent("PageLimit") {
            Text("\(pageLimit)")
        }

        Button {
            pageLimit += 1
        } label: {
            Text("Increment")
        }
    }
}
```

Left side of mutating operator isn't mutable: 'self' is immutable

Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

    init(key: String, defaultValue: T) {
        self.key = key
        self.defaultValue = defaultValue
    }

    init(wrappedValue: T, key: String) {
        self.defaultValue = wrappedValue
        self.key = key
    }

    var wrappedValue: T {
        get {
            UserDefaults.standard.value(forKey: key) as? T ?? defaultValue
        }
        set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

Property Wrapper

```
@propertyWrapper struct UserDefaultStore<T> {
    let key: String
    let defaultValue: T

    init(key: String, defaultValue: T) {
        self.key = key
        self.defaultValue = defaultValue
    }

    init(wrappedValue: T, key: String) {
        self.defaultValue = wrappedValue
        self.key = key
    }

    var wrappedValue: T {
        get {
            UserDefaults.standard.value(forKey: key) as? T ?? defaultValue
        }
        nonmutating set {
            UserDefaults.standard.setValue(newValue, forKey: key)
        }
    }
}
```

Property Wrapper

```
struct PostView: View {
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    var body: some View {
        LabeledContent("PageLimit") {
            Text("\(pageLimit)")
        }

        Button {
            pageLimit += 1
        } label: {
            Text("Increment")
        }
    }
}
```

Left side of mutat

Property Wrapper

```
struct PostView: View {
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    var body: some View {
        LabeledContent("PageLimit") {
            Text("\(pageLimit)")
        }

        Button {
            pageLimit += 1
        } label: {
            Text("Increment")
        }
    }
}
```

pageLimit is NOT updating when incremented

Property Wrapper

```
struct PostView: View {
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    var body: some View {
        LabeledContent("PageLimit") {
            Text("\(pageLimit)")
        }
        Button {
            pageLimit += 1
        } label: {
            Text("Increment")
        }
    }
}
```

SwiftUI does NOT observe the value change.

Property Wrapper

```
struct PostView: View {
    @UserDefaultStore(key: "pageLimit") var pageLimit: Int = 10

    var body: some View {
        LabeledContent("PageLimit") {
            Text("\(pageLimit)")
        }
        Button {
            pageLimit += 1
        } label: {
            Text("Increment")
        }
    }
}
```

SwiftUI does NOT observe the State change.

Attribute Graph

Other Side of SwiftUI

Private Framework*

Ratnesh Jain

=== AttributeGraph: cycle detected through attribute 290 ===
=== AttributeGraph: cycle detected through attribute 320 ===
=== AttributeGraph: cycle detected through attribute 360 ===
=== AttributeGraph: cycle detected through attribute 420 ===

/System/Library/PrivateFrameworks/AttributeGraph.framework

Attribute Graph

Other Side of SwiftUI

- Reads the View’s declaration.
- Creates a View Tree
- Keep track of State, Binding and other view dependency
- Creates a Persistent Reader Tree
- Observe the state, dependency changes
  - Throw away existing View Tree
  - Creates a new View tree
  - Re-render the Reader tree with smart difference.

Attribute Graph

Read View's Declaration

Opaque Type

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Read View's Declaration

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Opaque Type

- Type information is not directly visible to user/dev
- Compiler knows all type information

Attribute Graph

Read View’s Declaration

```
extension View {
    func mirror() {
        print(Mirror(reflecting: self).subjectType)
    }
}
```

```
Group<_ConditionalContent<ProgressView<EmptyView, EmptyView>,
ModifiedContent<VStack<TupleView<(ModifiedContent<ModifiedContent<Image,
_EnvironmentKeyWritingModifier<Scale>>, _ForegroundStyleModifier<TintShapeStyle>>, Text)>>, _PaddingLayout>>>
```

Attribute Graph

View Tree

```
Group<_ConditionalContent<ProgressView<EmptyView,
EmptyView>,
ModifiedContent<VStack<TupleView<(ModifiedContent
<ModifiedContent<Image,
_EnvironmentKeyWritingModifier<Scale>>,
_ForegroundStyleModifier<TintShapeStyle>>,
Text)>>, _PaddingLayout>>>
```

Attribute Graph

View Tree

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Keep Track of State and Dependencies

```swift
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Keep Track of State and Dependencies

```swift
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Keep Track of State and Dependencies

```swift
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Render Tree*

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Render Tree*

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Render Tree*

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Render Tree*

Attribute Graph

Render Tree*

Attribute Graph

Render Tree*

Attribute Graph

Render Tree*

```
struct ContentView: View {
    @State private var isLoading: Bool = false
    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: "globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, world!")
                }
                .padding()
            }
        }
    }
}
```

Attribute Graph

Render Tree*

Attribute Graph

Render Tree*

Attribute Graph

Other Side of SwiftUI

- Reads the View’s declaration.
- Creates a View Tree
- Keep track of State, Binding and other view dependency
- Creates a Persistent Reader Tree
- Observe the state, dependency changes
  - Throw away existing View Tree
  - Creates a new View tree
  - Re-render the Reader tree with smart difference.

Attribute Graph

For more details, please visit

https://chris.eidhof.nl/presentations/day-in-the-life/

Attribute Graph

For more details, please visit

A Day in a Life of SwiftUI View

https://chris.eidhof.nl/presentations/day-in-the-life/

Attribute Graph

Other Side of SwiftUI

- Reads the View’s declaration.
- Creates a View Tree
- Keep track of State, Binding and other view dependency
- Creates a Persistent Reader Tree
- Observe the state, dependency changes
  - Throw away existing View Tree
  - Creates a new View tree
  - Re-render the Reader tree with smart difference.

Attribute Graph

Other Side of SwiftUI

- Reads the View’s declaration.
- Creates a View Tree
- Keep track of State, Binding and other view dependency
- Creates a Persistent Reader Tree
- Observe the state, dependency changes
  - Throw away existing View Tree
  - Creates a new View tree
  - Re-render the Reader tree with smart difference.

State, Binding, Environment

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
@frozen @propertyWrapper public struct State<Value> : DynamicProperty {

    public init(wrappedValue value: Value)

    public init(initialValue value: Value)

    public var wrappedValue: Value { get nonmutating set }

    public var projectedValue: Binding<Value> { get }
}
```

State, Binding, Environment

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
@frozen @propertyWrapper @dynamicMemberLookup public struct Binding<Value> {

    public init(get: @escaping () -> Value, set: @escaping (Value) -> Void)

    public static func constant(_ value: Value) -> Binding<Value>

    public var wrappedValue: Value { get nonmutating set }

    public var projectedValue: Binding<Value> { get }

    public init(projectedValue: Binding<Value>)

    public subscript<Subject>(dynamicMember keyPath: WritableKeyPath<Value, Subject>) -> Binding<Subject> {
}

@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
extension Binding : DynamicProperty {}
```

State, Binding, Environment

```
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
@frozen @propertyWrapper public struct Environment<Value> : DynamicProperty {

    @inlinable public init(_ keyPath: KeyPath<EnvironmentValues, Value>)

    @inlinable public var wrappedValue: Value { get }
}
```

DynamicProperty

```swift
/// An interface for a stored variable that updates an external property of a
/// view.
///
/// The view gives values to these properties prior to recomputing the view's
/// ``View/body-swift.property``.
@available(iOS 13.0, macOS 10.15, tvOS 13.0, watchOS 6.0, *)
public protocol DynamicProperty {

    /// Updates the underlying value of the stored value.
    ///
    /// SwiftUI calls this function before rendering a view's
    /// ``View/body-swift.property`` to ensure the view has the most recent
    /// value.
    mutating func update()
}
```

DynamicProperty

An interface for a stored variable that updates an external property of a view

DynamicProperty

An interface for a stored variable that updates an external property of a view

The view gives values to these properties prior to recomputing the view's body property

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Ratnesh Jain

Attribute Graph

```swift
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Attribute Graph

```swift
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Ratnesh Jain

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Ratnesh Jain

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

Ratnesh Jain

Attribute Graph

```
struct ContentView: View {
    @State private var isLoading: Bool = false

    var body: some View {
        Group {
            if isLoading {
                ProgressView()
            } else {
                VStack {
                    Image(systemName: “globe")
                        .imageScale(.large)
                        .foregroundStyle(.tint)
                    Text("Hello, World")
                }
                .padding()
            }
        }
        .task {
            isLoading = true
            try? await Task.sleep(for: .seconds(2))
            isLoading = false
        }
    }
}
```

SwiftUI’s Magic

Attribute Graph

Ratnesh Jain

Let’s Study in

Thank You 🙇🏻‍♂️