Data Flow Through SwiftUI

Luca Bernardi, SwiftUI Engineer
Raj Ramamurthy, SwiftUI Engineer
Principles of Data Flow

Anatomy of an Update

Understanding Your Data
Principles of Data Flow
Anatomy of an Update
Understanding Your Data
Principles of Data Flow
Anatomy of an Update
Understanding Your Data
What is data?
Tools for Data Flow

@Environment
@Binding
@State

Property
BindableObject
Data Access as a Dependency

PlayerView

isPlaying: Bool
Data Access as a Dependency

PlayerView

isPlaying: Bool
Data Access as a Dependency

PlayerView -> false
Data Access as a Dependency

PlayerView

false
Data Access as a Dependency

PlayerView → true
Source of Truth
Source of Truth

- post: PhotoPost
-.isPlaying: Bool
- accentColor: Color
- store: MessageStore
Duplicated Source of Truth

View

View

isPlaying: Bool

isPlaying: Bool
Duplicated Source of Truth

```
View

View

isPlaying: Bool

View

isPlaying: Bool
```
Duplicated Source of Truth

isPlaying: Bool
Single Source of Truth

Diagram showing views connected with a flag `isPlaying: Bool`
Single Source of Truth

isPlaying: Bool
struct PlayerView : View {
    let episode: Episode

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
        }
    }
}
struct PlayerView : View {
    let episode: Episode

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
        }
    }
}
struct PlayerView : View {
    let episode: Episode

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
        }
    }
}
struct PlayerView : View {
    let episode: Episode

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
        }
    }
}
struct PlayerView : View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
struct PlayerView : View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView : View {
    let episode: Episode
    var isPlaying: Bool

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            Button(action: {
                self.isPlaying.toggle()
            }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
### Property Wrapper

Wraps property access with additional behavior

<table>
<thead>
<tr>
<th>What’s New in Swift</th>
<th>WWDC 2019</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modern Swift API Design</td>
<td>Thursday, 2:00</td>
</tr>
</tbody>
</table>
@State private var isPlaying: Bool = false
@State private var isPlaying: Bool = false
Anatomy of a View Update

PlayerView

- Text
- Text
- Button

SwiftUI

isPlaying = false
Anatomy of a View Update

PlayerView

- Text
- Text
- Button

SwiftUI

isPlaying = false
Anatomy of a View Update

PlayerView

- Text
- Text
- Button

SwiftUI

isPlaying = true
Anatomy of a View Update

PlayerView

Text
Text
Button

SwiftUI isPlaying = true
Anatomy of a View Update

- **PlayerView**
  - Text
  - Text
  - Button

SwiftUI:.isPlaying = true
Anatomy of a View Update

SwiftUI

PlayerView

- Text
- Text
- Button

isPlaying = true
Every `@State` is a source of truth.
Views are a function of state, not of a sequence of events.
User Interaction

Action
User Interaction

SwiftUI

Action
User Interaction

Action → State → View

Mutation

Updates
User Interaction → Action → State → View → Render → User Interaction

- Action
- View
- State
- Updates
- Mutation

SwiftUI
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: { self.isPlaying.toggle() }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: { self.isPlaying.toggle() }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            Button(action: { self.isPlaying.toggle() }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            Button(action: { self.isPlaying.toggle() }) {
                Image(systemName: isPlaying ? "pause.circle" : "play.circle")
            }
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            PlayButton()
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            PlayButton()
        }
    }
}
struct PlayButton : View {
    @State private var isPlaying: Bool = false

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
struct PlayButton: View {
    @State private var isPlaying: Bool = false

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
struct PlayButton : View {
  @State private var isPlaying: Bool = false

  var body: some View {
    Button(action: {
      self.isPlaying.toggle()
    }) {
      Image(systemName: isPlaying ? "pause.circle" : "play.circle")
    }
  }
}
@Binding Property Wrapper

Read and write without ownership

Derivable from @State

@Binding var isPlaying: Bool
struct PlayButton : View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
struct PlayButton : View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            PlayButton(isPlaying: $isPlaying)
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            PlayButton(isPlaying: $isPlaying)
        }
    }
}
struct PlayerView: View {
    let episode: Episode

    @State private var isPlaying: Bool = false

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            PlayButton(isPlaying: $isPlaying)
        }
    }
}
PlayerViewController
public struct Toggle<Label> : View {
    public init(
        isOn: Binding<Bool>,
        label: () -> Label
    )
}

public struct TextField : View {
    init(
        _ text: Binding<String>
    )
}

public struct Slider : View {
    public init<V : BinaryFloatingPoint>(
        value: Binding<V>
    )
}
Views are for Data

Layout
Visual Effects
Navigation
Gestures
Drawing
Animations
// Animated Changes

struct PlayButton : View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            self.isPlaying.toggle()
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
struct PlayButton : View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            withAnimation { self.isPlaying.toggle() }
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
// Animated Changes

struct PlayButton : View {
    @Binding var isPlaying: Bool

    var body: some View {
        Button(action: {
            withAnimation { self.isPlaying.toggle() }
        }) {
            Image(systemName: isPlaying ? "pause.circle" : "play.circle")
        }
    }
}
Working With External Data
Tools for Data Flow

BindableObject

Property

@Environment

@Binding

@State
SwiftUI

![SwiftUI logo](image-url)
Combine Publisher

Single abstraction

Main thread: use `.receive(on:)`

---

Introducing Combine and Advances in Foundation  Thursday, 10:00

Combine in Practice  Thursday, 2:00
struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = true
    @State private var currentTime: TimeInterval = 0.0

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)

            PlayButton(isPlaying: $isPlaying)

            Text("\\(currentTime, formatter: currentTimeFormatter)")
        }
    }
}
struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = true
    @State private var currentTime: TimeInterval = 0.0

    var body: some View {
        VStack {
            Text(episode.title).foregroundColor(isPlaying ? .white : .gray)
            Text(episode.showTitle).font(.caption).foregroundColor(.gray)
            PlayButton(isPlaying: $isPlaying)
            Text("\(currentTime, formatter: currentTimeFormatter)")
        }
    }
}
struct PlayerView : View {
    let episode: Episode
    @State private var isPlaying: Bool = true
    @State private var currentTime: TimeInterval = 0.0

    var body: some View {
        VStack {
            // ...
            Text("\((playhead, formatter: currentTimeFormatter)"")
        }
        .onReceive(PodcastPlayer.currentTimePublisher) { newCurrentTime in
            self.currentTime = newCurrentTime
        }
    }
}
struct PlayerView: View {
    let episode: Episode
    @State private var isPlaying: Bool = true
    @State private var currentTime: TimeInterval = 0.0

    var body: some View {
        VStack {
            // …
            Text("\(playhead, formatter: currentTimeFormatter)")
        }
        .onReceive(PodcastPlayer.currentTimePublisher) { newCurrentTime in
            self.currentTime = newCurrentTime
        }
    }
}
class PodcastPlayerStore {
    var currentTime: TimeInterval
    var isPlaying: Bool
    var currentEpisode: Episode

    func advance() {
    }
    func skipForward() {
    }
    func skipBackward() {
    }
}
class PodcastPlayerStore : BindableObject {

    var didChange = PassthroughSubject<Void, Never>()

    // …

    func advance() {
        currentEpisode = nextEpisode
        currentTime = 0.0
        // Notify subscribers that the player changed
        didChange.send()
    }
}
class PodcastPlayerStore : BindableObject {

    var didChange = PassthroughSubject<Void, Never>()

    // …

    func advance() {
        currentEpisode = nextEpisode
        currentTime = 0.0
        // Notify subscribers that the player changed
        didChange.send()
    }
}
Creating Dependencies on BindableObject
Creating Dependencies on BindableObject

Model -> @ObjectBinding -> View
Creating Dependencies on BindableObject

```swift
struct MyView : View {
    @ObjectBinding var model: MyModelObject
    ...
}

MyView(model: modelInstance)
```
Creating Dependencies on BindableObject

```swift
struct MyView : View {
    @ObjectBinding var model: MyModelObject
    ...
}

MyView(model: modelInstance)
```
Creating Dependencies on BindableObject

```swift
struct MyView: View {
    @ObjectBinding var model: MyModelObject
    ...
}

MyView(model: modelInstance)
```
Creating Dependencies on BindableObject

```swift
struct MyView : View {
    @ObjectBinding var model: MyModelObject
    ...
}

MyView(model: modelInstance)
```
Creating Dependencies on BindableObject

Pass directly with @ObjectBinding

Automatic dependency tracking

```swift
struct MyView : View {
    @ObjectBinding var model: MyModelObject
    ...
}
```

MyView(model: modelInstance)
Creating Dependencies Indirectly

Model

View

View

View
Creating Dependencies Indirectly

Model

Environment

Environment

Environment

View

View
Creating Dependencies Indirectly

Model

.environmentObject()

Environment

Environment

Environment

View

View

View
Creating Dependencies Indirectly

Model
  .environmentObject()

Environment

View
@EnvironmentObject

Environment

Environment

View
Creating Dependencies Indirectly

- Model
  - .environmentObject()
  - Environment
    - Environment
      - Environment
        - @EnvironmentObject
          - View
          - @EnvironmentObject
            - View
Creating Dependencies Indirectly

```swift
Model
    .environmentObject()
    .environmentObject()
    .environmentObject()

View
    @EnvironmentObject
    @EnvironmentObject
```
struct PlayerView: View {
    @EnvironmentObject var player: PodcastPlayerStore

    var body: some View {
        VStack {
            Text(player.currentEpisode.title)
                .foregroundColor(isPlaying ? .white : .gray)
            Text(player.currentEpisode.showTitle)
                .font(.caption).foregroundColor(.gray)
            PlayButton(isPlaying: $player.isPlaying)

            Text("\(player.currentTime, formatter: playheadTimeFormatter)")
        }
    }
}
struct PlayerView: View {
    @EnvironmentObject var player: PodcastPlayerStore

    var body: some View {
        VStack {
            Text(player.currentEpisode.title)
                .foregroundColor(isPlaying ? .white : .gray)
            Text(player.currentEpisode.showTitle)
                .font(.caption).foregroundColor(.gray)
            PlayButton(isPlaying: $player.isPlaying)
            Text("\(player.currentTime, formatter: playheadTimeFormatter)")
        }
    }
}
@ObjectBinding

Model

View
View
View
View
View
View
@ObjectBinding
@EnvironmentObject
@EnvironmentObject

Model -> Environment

View -> View -> View -> View
Environment

Data applicable to an entire hierarchy

Convenience for indirection

Accent color, right-to-left, and more
Sources of Truth

@State
- View-local Value
- Framework Managed

BindableObject
- External Reference
- Developer Managed
Building Reusable Components

Read-only: Swift property, Environment

Read-write: @Binding

Prefer immutable access
@Binding

First class reference to data

Great for reusability

Use $ to derive from source
Using State Effectively

Limit use if possible

Use derived `Binding` or value

Prefer `BindableObject` for persistence

Example: Button highlighting
Next Steps

Minimize sources of truth

Understand your data

Build a great app!
## More Information

developer.apple.com/wwdc19/226

<table>
<thead>
<tr>
<th>Session Title</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>SwiftUI Essentials</td>
<td>WWDC 2019</td>
</tr>
<tr>
<td>Integrating SwiftUI</td>
<td>Thursday, 3:00</td>
</tr>
<tr>
<td>Building Custom Views with SwiftUI</td>
<td>Friday, 9:00</td>
</tr>
</tbody>
</table>