Article

Observing playback state in SwiftUI

Keep your user interface in sync with state changes from playback objects.

Overview

An essential task when building custom media players is observing the state of playback objects to determine their readiness for playback and track other important lifecycle events. The way you typically do this is using key-value observing, but starting in iOS 26, tvOS 26, macOS 26, and visionOS 26, AVFoundation provides a new way to monitor playback state based on the Observation framework. AVFoundation supports this framework by making its core playback types conform to the `Observable` protocol. This means you can use `AVPlayer` or `AVQueuePlayer`, along with their associated `AVPlayerItem` and `AVPlayerItemTrack` objects to drive state changes directly within your SwiftUI views.

Opt-in to playback observation

AVFoundation supports monitoring playback state with Observation, but it doesn't enable this feature by default. Instead, you opt-in to this behavior by setting a `true` value for the `isObservationEnabled` property of the `AVPlayer` class.

```
// Opt-in to observing with the Observation framework.
AVPlayer.isObservationEnabled = true
```

Perform this opt-in early in your app lifecycle, such as in your main `App` structure or `UIApplicationDelegate` (in a mixed UIKit and SwiftUI app). This setting is global to your app and must be set *before* creating playback objects. Attempting to change its value after creating these objects results in AVFoundation throwing an exception.

Store playback state

You define a single source of truth in your app using a SwiftUI `State` variable. This property wrapper always instantiates its default value when SwiftUI creates a view. When using it to store playback objects, either directly or as part of a custom `@Observable` model object, avoid performance issues or other potential side effects by deferring the creation of these objects by using the `task(priority: :)` modifier. For example, in a simple playback case you could define a state variable to hold a player object and initialize it like shown below:

```
struct PlayerView: View {
    let url: URL
    // Don't create the player directly to avoid performance issues or other side ef
    @State private var player: AVPlayer?

    var body: some View {
        ZStack {
            if let player {
                VideoView(player: player)
                TransportView(player: player)
            } else {
                LoadingView()
            }
        }
        // Use the task modifier to defer creating the player to ensure
        // SwiftUI creates it only once when it first presents the view.
        .task {
            player = AVPlayer(url: url)
        }
    }
}
```

Using the task modifier ensures that SwiftUI initializes the player object only once when it first presents the view.

Observe state changes

Because the playback objects adopt the `Observable` protocol, you can use them directly within SwiftUI views like any other `@Observable` object. For example, you can pass a player instance to a view and have the view automatically redraw itself as playback state changes.

```
struct TransportView: View {

    // A player object passed from the view that owns the player reference.
    let player: AVPlayer

    // Observe the time control status property to determine whether playback is act
    private var isPlaying: Bool {
        player.timeControlStatus == .playing
    }

    var body: some View {
        // A button that toggles the state of playback.
        Button {
            if isPlaying {
                player.pause()
            } else {
                player.play()
            }
        } label: {
            Image(systemName: isPlaying ? "pause.fill" : "play.fill")
        }
        // Observe the player item's status property to determine when to enable the
        .disabled(player.currentItem?.status != .readyToPlay)
    }
}
```

The transport view defines an `isPlaying` property that uses the player object's `timeControlStatus` property to determine whether it's actively playing media. It uses this property to drive changes to the button's presentation and behavior. The view also observes the current player item and enables the button only after the item's `status` indicates it's ready to play.

> **Note**
>
> You can use Observation to monitor general state properties like a player's `currentItem` or `rate`, but it isn't suitable for observing continuous state changes like the current playback time. Instead, use the dedicated periodic and boundary time observation methods for this purpose. For more information, see Monitoring playback progress in your app.

See Also

Playback control

Controlling the transport behavior of a player

Play, pause, and seek through a media presentation.

{} Creating a seamless multiview playback experience

Build advanced multiview playback experiences with the AVFoundation and AVRouting frameworks.

`class AVPlayer`

An object that provides the interface to control the player's transport behavior.

`class AVPlayerItem`

An object that models the timing and presentation state of an asset during playback.

`class AVPlayerItemTrack`

An object that represents the presentation state of an asset track during playback.

`class AVQueuePlayer`

An object that plays a sequence of player items.

`class AVPlayerLooper`

An object that loops media content using a queue player.