Article

Monitoring playback progress in your app

Observe the playback of a media asset to update your app’s user-interface state.

Overview

Media playback apps commonly need to monitor playback progress to drive the state of player UI or perform other actions. Monitoring this state requires a higher level of time precision than key-value observing can deliver, so `AVPlayer` provides specific API to observe playback time. This article describes how you can observe this state at regular intervals or as playback crosses specific time boundaries.

Observe the current playback time at regular intervals

The most common way to observe a player’s current time is at regular intervals. Observing it this way is useful when driving the state of a time display in a player’s user interface.

To observe the player’s current time at regular intervals, call its `addPeriodicTimeObserver(forInterval:queue:using:)` method. This method takes a `CMTime` value that represents the interval at which to observe the time, a serial dispatch queue, and a callback that the player invokes at the specified time interval. The following example adds an observer that the player calls every half-second during normal playback:

```swift
@Published private(set) var duration: TimeInterval = 0.0
@Published private(set) var currentTime: TimeInterval = 0.0

private let player = AVPlayer()
private var timeObserver: Any?

/// Adds an observer of the player timing.
private func addPeriodicTimeObserver() {
    // Create a 0.5 second interval time.
    let interval = CMTime(value: 1, timescale: 2)
    timeObserver = player.addPeriodicTimeObserver(forInterval: interval,
                                                  queue: .main) { [weak self] time i
        guard let self else { return }
        // Update the published currentTime and duration values.
        currentTime = time.seconds
        duration = player.currentItem?.duration.seconds ?? 0.0
    }
}

/// Removes the time observer from the player.
private func removePeriodicTimeObserver() {
    guard let timeObserver else { return }
    player.removeTimeObserver(timeObserver)
    self.timeObserver = nil
}
```

Always pair a call to the player's `addPeriodicTimeObserver(forInterval:queue:using:)` method with a call to `removeTimeObserver(_:)` when you're finished monitoring the state. Failing to observe this rule results in undefined behavior.

Observe the playback of specific times within a media presentation

Another way to observe the player is when it crosses specific times boundaries during playback. You can respond to the passage of these times by updating your player UI or performing other actions.

To have the player notify your app as it cross specific points in the media timeline, call the player's `addBoundaryTimeObserver(forTimes:queue:using:)` method. This method takes an array of `NSValue` objects that wrap `CMTime` values that define your boundary times, a serial dispatch queue, and a callback closure. The following example shows how to define boundary times for each quarter of playback:

```
/// Adds an observer of the player traversing specific times during standard playbac
private func addBoundaryTimeObserver() async throws {

    // Asynchronously load the duration of the asset.
    let duration = try await asset.load(.duration)

    // Divide the asset's duration into quarters.
    let quarterDuration = CMTimeMultiplyByRatio(duration,
                                                multiplier: 1,
                                                divisor: 4)

    var currentTime = CMTime.zero
    var times = [NSValue]()

    // Calculate boundary times.
    while currentTime < duration {
        currentTime = currentTime + quarterDuration
        times.append(NSValue(time:currentTime))
    }

    timeObserver = player.addBoundaryTimeObserver(forTimes: times,
                                                  queue: .main) { [weak self] in
        // Update the percentage complete in the user interface.
    }
}
```

If you add either a periodic or boundary time observer, you need to remove observation by calling `removeTimeObserver(_:)` when complete.

See Also

Presentation

{} Using HEVC video with alpha

Play, write, and export HEVC video with an alpha channel to add overlay effects to your video processing.

`class AVPlayerLayer`

An object that presents the visual contents of a player object.

`class AVSynchronizedLayer`

A Core Animation layer that derives its timing from a player item so that you can synchronize layer animations with media playback.