Article

Controlling the transport behavior of a player

Play, pause, and seek through a media presentation.

Overview

AVFoundation provides comprehensive support for playing media assets, including local and remote file-based media and also media streamed with HTTP Live Streaming. The framework models its media assets using the `AVAsset` class, which provides a consistent interface to load and inspect your media, regardless of its type or location. Use an `AVPlayer` object to play media assets in the form of `AVPlayerItem` objects, which model the dynamic state of an asset such as its `currentTime()`.

Understanding how to effectively use `AVPlayer` is essential for anyone building a custom player UI or otherwise requiring programmatic control of playback.

Observe playback readiness

When you create a player item, it starts with a status of `AVPlayerItem.Status.unknown`, which means the system hasn't attempted to load its media for playback. Only when you associate the item with an `AVPlayer` object does the system begin loading an asset's media.

To know when the player item is ready for playback, observe the value of its `status` property. Add this observation before you call the player's `replaceCurrentItem(with:)` method, because associating the player item with a player is the system's cue to load the item's media:

```
func playMedia(at url: URL) {
    let asset = AVAsset(url: url)
    let playerItem = AVPlayerItem(
        asset: asset,
        automaticallyLoadedAssetKeys: [.tracks, .duration, .commonMetadata]
    )
    // Register to observe the status property before associating with player.
    playerItem.publisher(for: \.status)
        .removeDuplicates()
        .receive(on: DispatchQueue.main)
        .sink { [weak self] status in
            guard let self else { return }
            switch status {
            case .readyToPlay:
                // Ready to play. Present playback UI.
            case .failed:
                // A failure while loading media occurred.
            default:
                break
            }
        }
        .store(in: &subscriptions)

    // Set the item as the player's current item.
    player.replaceCurrentItem(with: playerItem)
}
```

When the player item reaches a `AVPlayerItem.Status.readyToPlay` state, present or enable your playback UI. Alternatively, if a failure occurs, show the appropriate status in the player.

Control the playback rate

A player provides the `play()` and `pause()` methods as its primary means of controlling its playback rate. When a player item is ready for playback, call the player's `play()` method to request that playback begins at the `defaultRate`, which has an initial value of `1.0` (the natural rate). By default, a player automatically waits to start playback until it has sufficient media data available to minimize stalling. You can determine whether a player is in a paused, waiting to play, or playing state by observing its `timeControlStatus` value:

```
@Published var isPlaying = false

private func observePlayingState() {
    player.publisher(for: \.timeControlStatus)
        .receive(on: DispatchQueue.main)
        .map { $0 == .playing }
        .assign(to: &$isPlaying)
}
```

Observe changes to the `rate` property by observing notifications of type `rateDidChangeNotification`. Observing this notification is similar to key-value observing the `rate` property, but provides additional information about the reason for the rate change. Retrieve the reason from the notification’s `userInfo` dictionary using the `rateDidChangeReasonKey` constant:

```
// Observe changes to the playback rate asynchronously.
private func observeRateChanges() async {
    let name = AVPlayer.rateDidChangeNotification
    for await notification in NotificationCenter.default.notifications(named: name)
        guard let reason = notification.userInfo?[AVPlayer.rateDidChangeReasonKey] a
            continue
        }
        switch reason {
        case .appBackgrounded:
            // The app transitioned to the background.
        case .audioSessionInterrupted:
            // The system interrupts the app's audio session.
        case .setRateCalled:
            // The app set the player's rate.
        case .setRateFailed:
            // An attempt to change the player's rate failed.
        default:
            break
        }
    }
}
```

Seek through the media timeline

You can seek through a media timeline in several ways using the methods of `AVPlayer` and `AVPlayerItem`. The most common way is to use the player’s `seek(to:)` method, passing it a destination `CMTime` value. Call this method in an asynchronous context:

```
// Handle time update request from user interface.
func seek(to timeInterval: TimeInterval) async {
    // Create a CMTime value for the passed in time interval.
    let time = CMTime(seconds: timeInterval, preferredTimescale: 600)
    await avPlayer.seek(to: time)
}
```

You can call this method a single time to seek to the location, but you can also call it continuously such as when you use a `Slider` view.

The `seek(to:)` method is a convenient way to quickly seek through your presentation, but it's tuned for speed rather than precision. This means the actual time to which the player seeks may differ slightly from the time you request. If you need to implement precise seeking behavior, use the `seek(to:toleranceBefore:toleranceAfter:)` method, which lets you indicate the tolerated amount of deviation from your target time (before and after). For example, if you need to provide sample-accurate seeking behavior, specify tolerance values of zero:

```
// Seek precisely to the specified time.
await avPlayer.seek(to: time, toleranceBefore: .zero, toleranceAfter: .zero)
```

See Also

Playback control

Observing playback state in SwiftUI

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

{} 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.