Article

Adopting Picture in Picture in a Custom Player

Add controls to your custom player user interface to invoke Picture in Picture (PiP) playback.

Overview

Add PiP playback to your custom player by using the AVKit framework's `AVPictureInPictureController` class. This class lets you implement the same PiP behavior found in `AVPlayerViewController` in your custom player.

Configure Audio Session and Background Modes

To participate with PiP in iOS and tvOS, customize your app's audio playback capabilities by configuring its audio session and background modes. For more information, see Configuring your app for media playback.

Update Your Custom Player User Interface

Begin by adding a user interface (UI) to your custom player interface to enable users to begin PiP playback. Make this UI consistent with the system default UI that `AVPlayerViewController` presents. Access the standard images for controlling PiP playback by using the `pictureInPictureButtonStartImage` and `pictureInPictureButtonStopImage` class properties of `AVPictureInPictureController`. These methods return system default images to present in your UI.

```
import AVKit
```

```
@IBOutlet weak var pipButton: UIButton!

override func viewDidLoad() {
    super.viewDidLoad()

    let startImage = AVPictureInPictureController.pictureInPictureButtonStartImage
    let stopImage = AVPictureInPictureController.pictureInPictureButtonStopImage

    pipButton.setImage(startImage, for: .normal)
    pipButton.setImage(stopImage, for: .selected)
}
```

Use key-value observing (KVO) on the controller's `canStopPictureInPicture` property to display the appropriate affordances and provide the correct behavior in your playback UI. If `false`, display a start PiP affordance. If `true`, your app stops your custom playback UI and displays UI affordances to swap if you're creating a tvOS app. For more information about KVO, see `Using Key-Value Observing in Swift`.

Create the PiP Controller

Create an instance of `AVPictureInPictureController` to control PiP playback in your app. Before attempting to create the controller instance, verify that the current hardware supports PiP playback by calling the `isPictureInPictureSupported()` method.

```
var pipController: AVPictureInPictureController!
var pipPossibleObservation: NSKeyValueObservation?

func setupPictureInPicture() {
    // Ensure PiP is supported by current device.
    if AVPictureInPictureController.isPictureInPictureSupported() {
        // Create a new controller, passing the reference to the AVPlayerLayer.
        pipController = AVPictureInPictureController(playerLayer: playerLayer)
        pipController.delegate = self

        pipPossibleObservation = pipController.observe(\AVPictureInPictureControlle
options: [.initial, .new]) { [weak self] _, change in
            // Update the PiP button's enabled state.
            self?.pipButton.isEnabled = change.newValue ?? false
        }
    } else {
        // PiP isn't supported by the current device. Disable the PiP button.
        pipButton.isEnabled = false
```

```
    }
}
```

This example creates a new `AVPictureInPictureController` instance, passing it a reference to the `AVPlayerLayer` that presents the video content. The system supports displaying content from an `AVPlayerLayer` or `AVSampleBufferDisplayLayer` in a PiP window.

For PiP functionality to work, maintain a strong reference to the controller object.

> **Note**
>
> The PiP display doesn't use the `AVPlayerLayer` that you passed to `AVPictureInPictureController`, so AVFoundation stops vending video frames to `AVPlayerLayer` when PiP mode is active.

To participate in PiP life-cycle events, your code should adopt the `AVPictureInPictureControllerDelegate` protocol and set itself as the controller's delegate. Also, use KVO on the controller's `isPictureInPicturePossible` property to observe whether using PiP mode is possible in the current context, for example, when the system is displaying an active FaceTime window. By observing this property, you can determine when it's appropriate to change the enabled state of your PiP button.

Publish the Now Playing State

On tvOS, `MPNowPlayingSession` ties your `AVPlayer` instances to a session. Your app can have many playing sessions, and in the case of PiP, your player must be tied to a session. You can have a Now Playing session for your PiP content and one for your full-screen content. When you update a session, the system ignores updates from the default `MPNowPlayingInfoCenter`, so migrate away from `MPNowPlayingInfoCenter.default()` and switch to `MPNowPlayingSession` across your whole app.

The system determines which Now Playing information to display, so publish your information even if the UI isn't displaying your session — the system might display your session at any moment. For more information about Now Playing metadata, see the Now Playing Metadata Properties topic group at `MPNowPlayingInfoCenter`.

```
func publishNowPlayingMetadata() {
    nowPlayingSession.nowPlayingInfoCenter.nowPlayingInfo = nowPlayingInfo
    nowPlayingSession.becomeActiveIfPossible()
}
```

Handle User-Initiated Requests

With the `AVPictureInPictureController` setup complete, add an `@IBAction` method to handle user-initiated requests to start or stop PiP playback.

```
@IBAction func togglePictureInPictureMode(_ sender: UIButton) {
    if pipController.isPictureInPictureActive {
        pipController.stopPictureInPicture()
    } else {
        pipController.startPictureInPicture()
    }
}
```

> **Important**
>
> Only begin PiP playback in response to user interaction and never programmatically. The App Store review team rejects apps that fail to follow this requirement.

Restore Control to Your App

A user selects the stop PiP affordance in the PiP window to return control to your app. By default, this action terminates playback when control returns to the app. It's your responsibility to properly restore your video playback interface.

To handle the restore process, implement the `pictureInPictureController(_:restoreUserInterfaceForPictureInPictureStopWithCompletionHandler:)` delegate method and restore your player interface as needed. When the restoration is complete, call the completion handler with a value of `true`.

```
func pictureInPictureController(_ pictureInPictureController: AVPictureInPictureCont
                                restoreUserInterfaceForPictureInPictureStopWithCompl
    // Restore the user interface.
    completionHandler(true)
}
```

Dismiss Playback Controls

While PiP is active, dismiss playback controls in your main player, and present artwork in the PiP window to indicate that PiP mode is active. To implement this functionality, use the `pictureInPictureControllerWillStartPictureInPicture(_:)` and `pictureInPicture`

`ControllerDidStopPictureInPicture(_:)` delegate methods, and take the required actions.

```
func pictureInPictureControllerWillStartPictureInPicture(_ pictureInPictureControlle
    // Hide the playback controls.
    // Show the placeholder artwork.
}

func pictureInPictureControllerDidStopPictureInPicture(_ pictureInPictureController:
    // Hide the placeholder artwork.
    // Show the playback controls.
}
```

See Also

Picture in Picture

{} Adopting Picture in Picture Playback in tvOS

Add advanced multitasking capabilities to your video apps by using Picture in Picture playback in tvOS.

Adopting Picture in Picture in a Standard Player

Add Picture in Picture (PiP) playback to your app using a player view controller.

Adopting Picture in Picture for video calls

Add multitasking capability to your video-call apps by using Picture in Picture (PiP).

Accessing the camera while multitasking on iPad

Operate the camera in Split View, Slide Over, Picture in Picture, and Stage Manager modes.

`class AVPictureInPictureController`

A controller that responds to user-initiated Picture in Picture playback of video in a floating, resizable window.