

[AVFoundation](#) / [Capture setup](#) / Supporting Continuity Camera in your macOS app

Sample Code

Supporting Continuity Camera in your macOS app

Enable high-quality photo and video capture by using an iPhone camera as an external capture device.

Download

macOS 13.0+ | Xcode 16.0+

Overview

Continuity Camera brings the power of an iPhone device's camera and image signal processing to the Mac. It lets you use the rear-facing, wide-angle camera of iPhone to support high-quality photo and video capture in your macOS app. Continuity Camera also brings advanced features like Center Stage, Portrait mode, and Studio Light to all Mac devices. An important part of adopting this feature in your app is supporting automatic camera selection.

Starting in macOS 13, the operating system remembers the camera that it prefers to use for capture. It bases its preference on factors including capture quality, device positioning, and user preference. Apps observe the state of the system-preferred camera, and automatically update their camera selection as the value changes.

The sample app shows you how to access the iPhone camera and microphone, adopt automatic camera selection, and control and observe the state of system video effects.

Note

This sample code project is associated with WWDC22 session [10018: Bring Continuity Camera to your macOS app](#)

Configure the sample code project

To run this sample app, you'll need the following:

- A Mac with macOS 13 beta or later.
- An iPhone with iOS 16 beta or later.
- Xcode 14 beta or later.
- Both devices must be signed into an Apple ID account that uses two-factor authentication.

Connect the iPhone to the Mac over USB. The first time you run this sample, the system prompts you to grant the app access to the camera and microphone. You must allow the sample app to access both devices for it to function correctly.

Configure device discovery

When the app performs its initial setup, it configures two instances of AVCaptureDevice .DiscoverySession: one to discover video devices, and the other to discover audio devices. It initializes each discovery session with a list of devices that includes a built-in device, and also an externalUnknown device. Specifying an external unknown device type enables the discovery session to find compatible iPhone cameras and microphones, as well as supported capture devices from other vendors.

```
// Observe device cameras. Specify `.externalUnknown` to access an iPhone camera as
videoDiscoverySession = AVCaptureDevice.DiscoverySession(deviceTypes: [.builtInWideAreaCamera,
                                                                    .externalUnknown],
                                                            mediaType: .video,
                                                            position: .unspecified)

// Observe device microphones. Specify `.externalUnknown` to access an iPhone microphone
audioDiscoverySession = AVCaptureDevice.DiscoverySession(deviceTypes: [.builtInMicrophone,
                                                                    .externalUnknown],
                                                            mediaType: .audio,
                                                            position: .unspecified)
```

The app observes the state of each discovery session's devices array, and as devices connect and disconnect from the system, it updates the list of devices it presents in the user interface.

Set the initial video device selection

The sample app enables automatic camera selection by default, so when it performs its initial capture session configuration, it uses the system's preferred camera as its default camera device as shown below.

```
private var defaultVideoCaptureDevice: AVCaptureDevice {
    get throws {
        // Access the system's preferred camera.
        if let device = AVCaptureDevice.systemPreferredCamera {
            return device
        } else {
            // No camera is available on the host system.
            throw Error.noVideoDeviceAvailable
        }
    }
}
```

The `systemPreferredCamera` property of `AVCaptureDevice` provides an optional capture device value. However, it always provides a valid capture device unless the host system provides no built-in or connected devices.

Respond to system-preferred camera changes

The system-preferred camera changes based on device availability and user camera selection. The sample app provides a `PreferredCameraObserver` object that key-value observes the `systemPreferredCamera` property to monitor changes. When it observes a change to the value, it updates the state of its `@Published` property value as shown below.

```
class PreferredCameraObserver: NSObject, ObservableObject {

    private let systemPreferredKeyPath = "systemPreferredCamera"

    @Published private(set) var systemPreferredCamera: AVCaptureDevice?

    override init() {
        super.init()
        // Key-value observe the `systemPreferredCamera` class property on `AVCaptureDevice`.
        AVCaptureDevice.self.addObserver(self, forKeyPath: systemPreferredKeyPath, options: NSKeyValueObservingOptions(), context: nil)
    }

    override func observeValue(forKeyPath keyPath: String?, of object: Any?, change: [AnyHashable: Any]?, context: Any?) {
        switch keyPath {
        case systemPreferredKeyPath:
            // Update the observer's system-preferred camera value.
            systemPreferredCamera = change?["newKey"] as? AVCaptureDevice
        default:
            break
        }
    }
}
```

```

        super.observeValue(forKeyPath: keyPath, of: object, change: change, context: context) {
            // ...
        }
    }
}

```

If the app has automatic camera selection enabled, when it observes a change to the system-preferred camera, it automatically switches to the new device.

```

// The app calls this method when the system-preferred camera changes.
private func systemPreferredCameraChanged(to captureDevice: AVCaptureDevice) async {

    // If the SPC changes due to a device disconnection, reset the app
    // to its default device selections.
    guard isActiveVideoInputDeviceConnected else {
        resetToDefaultDevices()
        return
    }

    // If the "Automatic Camera Selection" checkbox is in an enabled state,
    // automatically select the new capture device.
    if isAutomaticCameraSelectionEnabled {
        await selectCaptureDevice(captureDevice)
    }
}

```

Update the user-preferred camera selection

When the app selects a new device, it removes the old device input, attempts to add an input for the new device, and if it succeeds, updates the state of the appropriate active input device as shown below.

```

// Remove the current input from the session.
session.removeInput(currentInput)

// Attempt to add the new device to the capture session.
let newInput = try addInput(for: device)

// Camera
if mediaType == .video {
    activeVideoInput = newInput
    if isUserSelection {

```

```

        // If the device change is due to user selection, set the UPC value,
        // which updates the state of the system-preferred camera.
        AVCaptureDevice.userPreferredCamera = device
    }
}
// Microphone
else {
    activeAudioInput = newInput
}
}

```

In the case of a camera device, if the selection request came from user input, the app also updates the state of the `userPreferredCamera`, which persists the user's preferred camera selection across app and device restarts.

Note

Setting the user-preferred camera also updates the value of the `[systemPreferredCamera][7]` property.

Support system video effects

When the app's selected camera changes, the system evaluates the new capture device's `activeFormat` to determine if it supports Center Stage. If it does, the app enables the Center Stage checkbox so you can change its value. Toggling the state of the feature sets the `centerStageControlMode` to `AVCaptureDevice.CenterStageControlMode.cooperative` and updates its enabled state.

```

@Published var isCenterStageEnabled = false {
    didSet {
        guard isCenterStageEnabled != AVCaptureDevice.isCenterStageEnabled else { return }
        AVCaptureDevice.centerStageControlMode = .cooperative
        AVCaptureDevice.isCenterStageEnabled = isCenterStageEnabled
    }
}

```

You can also enable Center Stage, along with video effects like Portrait mode and Studio Light, in Control Center. Because the enabled state of each effect can change externally, the app also key-value observes the state of these effects. Similar to how the app observes changes to the system-preferred camera, the app key-value observes the state of the `isCenterStageEnabled`, `isPortraitEffectEnabled`, and `isStudioLightEnabled` properties and updates the user interface state as the values change.

```
AVCaptureDevice.self.addObserver(self, forKeyPath: centerStageKeyPath, options: [.new], context: nil)
AVCaptureDevice.self.addObserver(self, forKeyPath: portraitEffectKeyPath, options: [.new], context: nil)
AVCaptureDevice.self.addObserver(self, forKeyPath: studioLightKeyPath, options: [.new], context: nil)
```

See Also

Continuity Camera

`{}` Supporting Continuity Camera in your tvOS app

Capture high-quality photos, video, and audio in your Apple TV app by connecting an iPhone or iPad as a continuity device.

`class` AVCaptureDeskViewApplication

An object that programmatically presents Desk View.