

[visionOS](#) / Displaying video from connected devices

Sample Code

Displaying video from connected devices

Show video from devices connected with the Developer Strap in your visionOS app.

Download

visionOS 2.2+ | Xcode 16.2+

Overview

Apple’s audiovisual frameworks allow your visionOS app to access video from USB video class (UVC) devices connected with the [Developer Strap](#) for Apple Vision Pro. You can use this functionality to display realtime video in your app. For example, a medical researcher can view the output from an endoscopic camera during a procedure. This article outlines the requirements to access UVC devices in visionOS, while the sample code project shows a picker for every device connected to Vision Pro and displays the selected device’s video feed.

Configure the sample code project

In the Xcode project, replace `Enterprise.license` with your license file. The sample app requires a valid license file to display the selected video feed.

Request the entitlement

UVC device access is a part of enterprise APIs for visionOS, a collection of APIs that unlock capabilities for enterprise customers. To use UVC device access, apply for the [UVC Device Access on visionOS](#) entitlement. For more information, including how to apply for this entitlement, see [Building spatial experiences for business apps with enterprise APIs for visionOS](#).

Add usage descriptions for camera access

To help protect people's privacy, visionOS limits app access to cameras and other sensors in Apple Vision Pro. You need to add an [NSCameraUsageDescription](#) to your app's information property list to provide a usage description that explains how your app uses the data these sensors provide. People see this description when your app prompts for access to camera data.

Create the device picker

Use an [AVCaptureDevice.DiscoverySession](#) to obtain an array of connected devices.

```
// ConnectionManager

private let discoverySession = AVCaptureDevice.DiscoverySession(deviceTypes: [.external],
                                                                mediaType: .video,
                                                                position: .unspecified)

private func updateDeviceList() {
    // Transform the `AVCaptureDevice` instances.
    let devices = discoverySession
        .devices
        .map { Device(id: $0.uniqueID, name: $0.localizedName) }

    ...
}
```

Next, observe [wasConnectedNotification](#) and [wasDisconnectedNotification](#) to update the array when a device connects or disconnects.

```
// ConnectionManager

private func observeDeviceConnectionStates() {
    Task {
        // Await notification of the system connecting a new device.
        for await _ in NotificationCenter.default.notifications(named: AVCaptureDevice.wasConnectedNotification) {
            updateDeviceList()
        }
    }

    Task {
        // Await notification of the system disconnecting a device.
        for await _ in NotificationCenter.default.notifications(named: AVCaptureDevice.wasDisconnectedNotification) {
            updateDeviceList()
        }
    }
}
```

```

        for await _ in NotificationCenter.default.notifications(named: AVCaptureDevic
            updateDeviceList()
        }
    }
}

```

Render a picker with an option for each device:

```

// ContentView

Picker("Device Picker", selection: $previewManager.selectedDevice) {
    Text("Select Device").tag(nil as Device?)
    ForEach(devices) {
        Text($0.name).tag($0)
    }
}

```

Display the selected device's video feed

Configure an AVCaptureSession to capture AVCaptureDeviceInput from the selected device and output it to an AVCaptureVideoDataOutput.

```

// CaptureManager

private let captureSession = AVCaptureSession()
private let videoDataOutput = AVCaptureVideoDataOutput()

...

private func setUpSession() {
    // Bracket the following configuration in a begin/commit configuration pair.
    captureSession.beginConfiguration()
    defer { captureSession.commitConfiguration() }

    // Drop frames that aren't rendered in a timely manner.
    videoDataOutput.alwaysDiscardsLateVideoFrames = true
    videoDataOutput.setSampleBufferDelegate(self, queue: sessionQueue)

    if captureSession.canAddOutput(videoDataOutput) {
        captureSession.addOutput(videoDataOutput)
    } else {

```

```

        assertionFailure("Unable to add video data output to capture session.")
    }
}

/// Stops capture from the previously selected device and, if provided, begins capture
/// - Parameter device: The device to capture video from or nil to stop capture all
func select(device: Device?) {
    // Bracket the following configuration in a begin/commit configuration pair.
    captureSession.beginConfiguration()
    defer { captureSession.commitConfiguration() }

    // Remove previous input, if one exists.
    for input in captureSession.inputs {
        captureSession.removeInput(input)
    }

    // Prepare the renderer to receive content from a new device.
    videoRenderer.flush(removingDisplayedImage: true)

    // Return early if the passed device is nil.
    guard let captureDevice = device?.captureDevice else { return }

    do {
        let authorizationStatus = AVCaptureDevice.authorizationStatus(for: .video)

        /// In the context of this sample, this check should always pass because `Co
        /// displays a message and terminates when the system denies access to the c
        precondition(authorizationStatus == .authorized,
            "Camera authorization is required to set up a device capture

        let input = try AVCaptureDeviceInput(device: captureDevice)

        // Add the new input, if possible.
        if captureSession.canAddInput(input) {
            captureSession.addInput(input)
        } else {
            assertionFailure("The input can't be added to the capture session.")
        }
    } catch {
        fatalError("Unable to create input for device. \(error)")
    }
}

```

Call `startRunning()` on the capture session to start the flow of data from the capture session's inputs to its outputs.

```
// CaptureManager

/// Begin the flow of data from the capture session's inputs to its outputs.
func start() {
    captureSession.startRunning()
}
```

`AVCaptureSession` delivers a steady stream of updates to the `AVCaptureVideoDataOutputSampleBufferDelegate` assigned to the `AVCaptureVideoDataOutput`. Each update includes a `CMSampleBuffer` that contains the latest video frame from the device. Render the `CMSampleBuffer` to an `AVSampleBufferDisplayLayer` using the layer's `AVSampleBufferVideoRenderer`.

```
// CaptureManager

/// The video renderer owned by the `AVSampleBufferDisplayLayer`
/// this app uses to display video to people.
nonisolated private let videoRenderer: AVSampleBufferVideoRenderer

...

extension CaptureManager: AVCaptureVideoDataOutputSampleBufferDelegate {
    nonisolated func captureOutput(_ output: AVCaptureOutput, didOutput sampleBuffer: CMSampleBuffer) {

        // If the renderer is ready for more data, queue the the sample buffer for rendering
        if videoRenderer.isReadyForMoreMediaData {
            videoRenderer.enqueue(sampleBuffer)
        }
    }
}
```

Add the `AVSampleBufferDisplayLayer` to a `UIView` and use a `UIViewRepresentable` to display the `UIView` in a SwiftUI view.

```
struct DevicePreview: UIViewRepresentable {
    /*
     In this sample, `preview` is an instance of `AVSampleBufferDisplayLayer`.
     `AVCaptureVideoDataOutputSampleBufferDelegate.captureOutput`
    */
}
```

```

    uses the layer's `sampleBufferRenderer` to enqueue the provided
    `CMSampleBuffer` for rendering.
    */
private let preview: CALayer

init(preview: CALayer) {
    self.preview = preview
}

func makeUIView(context: Context) -> SampleBufferPreview {
    SampleBufferPreview(preview: preview)
}

func updateUIView(_ previewView: SampleBufferPreview, context: Context) {
    // No-op.
}

class SampleBufferPreview: UIView {

    let preview: CALayer

    init(preview: CALayer) {
        self.preview = preview
        super.init(frame: .zero)
        layer.addSublayer(preview)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    override func layoutSubviews() {
        preview.frame = bounds
    }
}
}


```


Display an error when denying access to the camera


If the person hasn't granted camera access, the sample app prompts people to grant access in the Settings app. For more information about providing camera access in your app, see [Requesting authorization to capture and save media](#).

See Also

Enterprise APIs for visionOS

 Accessing the main camera
Add camera-based features to enterprise apps.

 Building spatial experiences for business apps with enterprise APIs for visionOS
Grant enhanced sensor access and increased platform control to your visionOS app by using entitlements.

 Locating and decoding barcodes in 3D space
Create engaging, hands-free experiences based on barcodes in a person's surroundings.