Sample Code

Capturing screen content in macOS

Stream desktop content like displays, apps, and windows by adopting screen capture in your app.

Download

macOS 15.0+ | Xcode 16.0+

Overview

This sample shows how to add high-performance screen capture to your Mac app by using ScreenCaptureKit. The sample explores how to create content filters to capture the displays, apps, and windows you choose. It then shows how to configure your stream output, retrieve video frames and audio samples, and update a running stream.

> **Note**
>
> This sample code project is associated with WWDC24 session 10088: Capture HDR content with ScreenCaptureKit.

Configure the sample code project

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

- A Mac with macOS 15 or later
- Xcode 16 or later

The first time you run this sample, the system prompts you to grant the app Screen Recording permission. After you grant permission, you need to restart the app to enable capture.

Create a content filter

Displays, running apps, and windows are the shareable content on a device. The sample uses the `SCShareableContent` class to retrieve the items in the form of `SCDisplay`, `SCRunningApplication`, and `SCWindow` objects respectively.

```
// Retrieve the available screen content to capture.
let availableContent = try await SCShareableContent.excludingDesktopWindows(false,
                                                                            onScreer
```

Before the sample begins capture, it creates an `SCContentFilter` object to specify the content to capture. The sample provides two options that allow for capturing either a single window or an entire display. When the capture type is set to capture a window, the app creates a content filter that only includes that window.

```
// Create a content filter that includes a single window.
filter = SCContentFilter(desktopIndependentWindow: window)
```

When a user specifies to capture the entire display, the sample creates a filter to capture only content from the main display. To illustrate filtering a running app, the sample contains a toggle to specify whether to exclude the sample app from the stream.

```
var excludedApps = [SCRunningApplication]()
// If a user chooses to exclude the app from the stream,
// exclude it by matching its bundle identifier.
if isAppExcluded {
    excludedApps = availableApps.filter { app in
        Bundle.main.bundleIdentifier == app.bundleIdentifier
    }
}
// Create a content filter with excluded apps.
filter = SCContentFilter(display: display,
                         excludingApplications: excludedApps,
                         exceptingWindows: [])
```

Create a stream configuration

An `SCStreamConfiguration` object provides properties to configure the stream’s output size, pixel format, audio capture settings, and more. The app’s configuration throttles frame updates to 60 fps, and configures the number of frames to keep in the queue at 5. Specifying more frames uses more memory, but may allow for processing frame data without stalling the display stream. The default value is 3 and shouldn't exceed 8 frames.

```swift
var streamConfig = SCStreamConfiguration()

if let dynamicRangePreset = selectedDynamicRangePreset?.scDynamicRangePreset {
    streamConfig = SCStreamConfiguration(preset: dynamicRangePreset)
}

// Configure audio capture.
streamConfig.capturesAudio = isAudioCaptureEnabled
streamConfig.excludesCurrentProcessAudio = isAppAudioExcluded
streamConfig.captureMicrophone = isMicCaptureEnabled

// Configure the display content width and height.
if captureType == .display, let display = selectedDisplay {
    streamConfig.width = display.width * scaleFactor
    streamConfig.height = display.height * scaleFactor
}

// Configure the window content width and height.
if captureType == .window, let window = selectedWindow {
    streamConfig.width = Int(window.frame.width) * 2
    streamConfig.height = Int(window.frame.height) * 2
}

// Set the capture interval at 60 fps.
streamConfig.minimumFrameInterval = CMTime(value: 1, timescale: 60)

// Increase the depth of the frame queue to ensure high fps at the expense of increa
// the memory footprint of WindowServer.
streamConfig.queueDepth = 5
```

Start the capture session

The sample uses the content filter and stream configuration to initialize a new instance of SCStream. To retrieve audio and video sample data, the app adds stream outputs that capture media of the specified type. When the stream captures new sample buffers, it delivers them to its stream output object on the indicated dispatch queues.

```
stream = SCStream(filter: filter, configuration: configuration, delegate: streamOutp

// Add a stream output to capture screen content.
try stream?.addStreamOutput(streamOutput, type: .screen, sampleHandlerQueue: videoSa
try stream?.addStreamOutput(streamOutput, type: .audio, sampleHandlerQueue: audioSam
try stream?.addStreamOutput(streamOutput, type: .microphone, sampleHandlerQueue: mic
stream?.startCapture()
```

After the stream starts, further changes to its configuration and content filter don't require restarting it. Instead, after you update the capture configuration in the user interface, the sample creates new stream configuration and content filter objects and applies them to the running stream to update its state.

```
try await stream?.updateConfiguration(configuration)
try await stream?.updateContentFilter(filter)
```

Process the output

When a stream captures a new audio or video sample buffer, it calls the stream output's `stream(_:didOutputSampleBuffer:of:)` method, passing it the captured data and an indicator of its type. The stream output evaluates and processes the sample buffer as shown below.

```
func stream(_ stream: SCStream, didOutputSampleBuffer sampleBuffer: CMSampleBuffer,

    // Return early if the sample buffer is invalid.
    guard sampleBuffer.isValid else { return }

    // Determine which type of data the sample buffer contains.
    switch outputType {
    case .screen:
        // Process the screen content.
    case .audio:
        // Process the audio content.
    }
}
```

Process a video sample buffer

If the sample buffer contains video data, it retrieves the sample buffer attachments that describe the output video frame.

```
// Retrieve the array of metadata attachments from the sample buffer.
guard let attachmentsArray = CMSampleBufferGetSampleAttachmentsArray(sampleBuffer,
                                                                     createIfNecessa
      let attachments = attachmentsArray.first else { return nil }
```

An SCStreamFrameInfo structure defines dictionary keys that the sample uses to retrieve metadata attached to a sample buffer. Metadata includes information about the frame's display time, scale factor, status, and more. To determine whether a frame is available for processing, the sample inspects the status for SCFrameStatus.complete.

```
// Validate the status of the frame. If it isn't `.complete`, return nil.
guard let statusRawValue = attachments[SCStreamFrameInfo.status] as? Int,
      let status = SCFrameStatus(rawValue: statusRawValue),
      status == .complete else { return nil }
```

The sample buffer wraps a CVPixelBuffer that's backed by an IOSurface. The sample casts the surface reference to an IOSurface that it later sets as the layer content of an NSView.

```
// Get the pixel buffer that contains the image data.
guard let pixelBuffer = sampleBuffer.imageBuffer else { return nil }

// Get the backing IOSurface.
guard let surfaceRef = CVPixelBufferGetIOSurface(pixelBuffer)?.takeUnretainedValue()
let surface = unsafeBitCast(surfaceRef, to: IOSurface.self)

// swiftlint:disable force_cast
// Retrieve the content rectangle, scale, and scale factor.
guard let contentRectDict = attachments[.contentRect],
      let contentRect = CGRect(dictionaryRepresentation: contentRectDict as! CFDicti
      let contentScale = attachments[.contentScale] as? CGFloat,
      let scaleFactor = attachments[.scaleFactor] as? CGFloat else { return nil }

// Create a new frame with the relevant data.
let frame = CapturedFrame(surface: surface,
                          contentRect: contentRect,
                          contentScale: contentScale,
                          scaleFactor: scaleFactor)
```

Process an audio sample buffer

If the sample buffer contains audio, it retrieves the data as an `AudioBufferList` as shown below.

```
private func handleAudio(for buffer: CMSampleBuffer) -> Void? {
    // Create an AVAudioPCMBuffer from an audio sample buffer.
    try? buffer.withAudioBufferList { audioBufferList, blockBuffer in
        guard let description = buffer.formatDescription?.audioStreamBasicDescriptio
              let format = AVAudioFormat(standardFormatWithSampleRate: description.m
              let samples = AVAudioPCMBuffer(pcmFormat: format, bufferListNoCopy: au
        else { return }
        pcmBufferHandler?(samples)
    }
}
```

The app retrieves the audio stream basic description that it uses to create an `AVAudioFormat`. It then uses the format and the audio buffer list to create a new instance of `AVAudioPCMBuffer`. If you enable audio capture in the user interface, the sample uses the buffer to calculate average levels for the captured audio to display in a simple level meter.

See Also

Essentials

ScreenCaptureKit updates

Learn about important changes to ScreenCaptureKit.

`Persistent Content Capture`

A Boolean value that indicates whether a Virtual Network Computing (VNC) app needs persistent access to screen capture.