

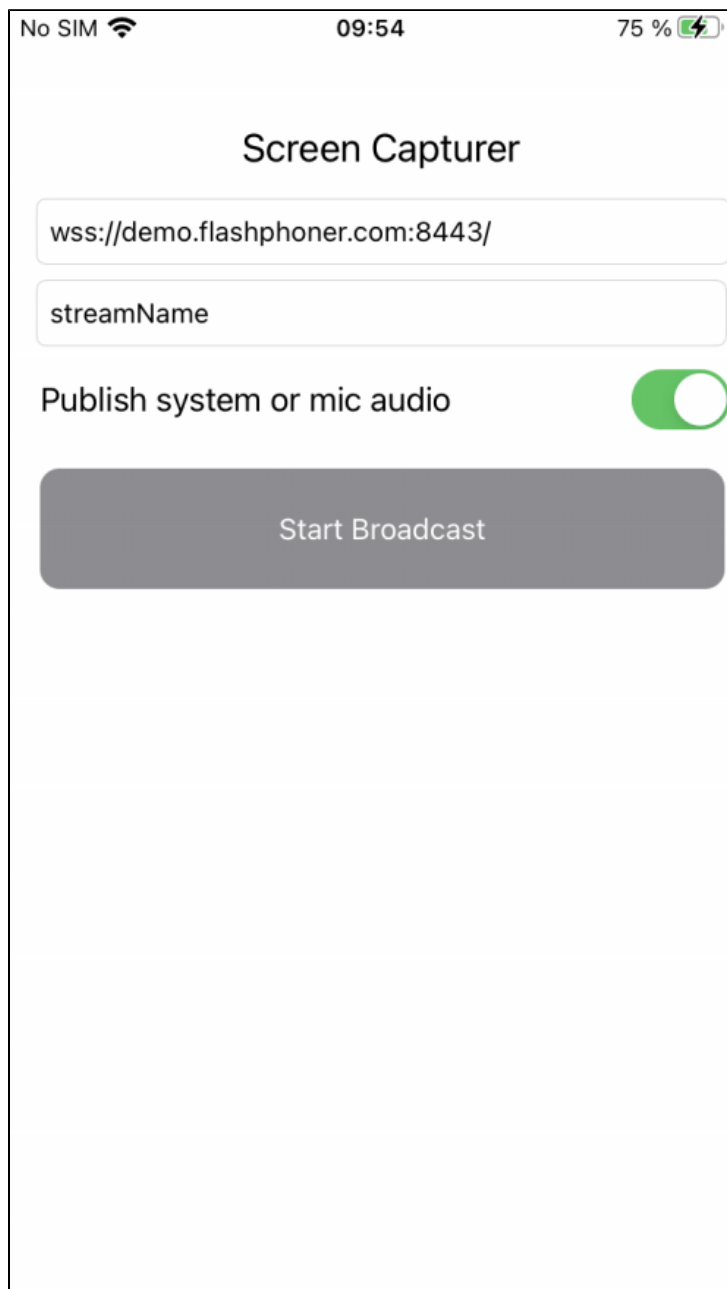
iOS Screen Capturer Swift

iOS application example to capture stream from device screen

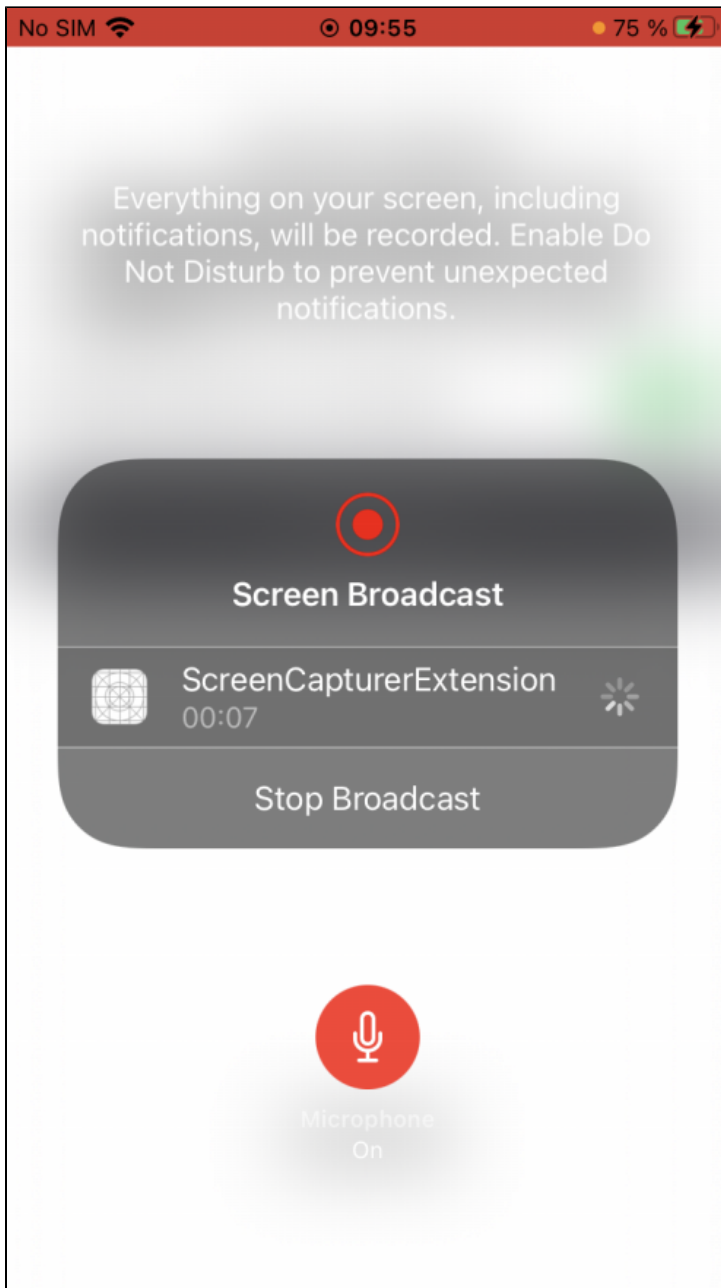
This example can be used to publish WebRTC stream from device screen with system audio or microphone audio capturing. The example works with iOS SDK [2.6.82](#) and newer.

The main application view is shown below. Inputs:

- WCS Websocket URL
- screen video stream name to publish

The image shows a screenshot of an iOS application titled "Screen Capturer". At the top, the status bar displays "No SIM", a Wi-Fi signal icon, the time "09:54", and a battery level of "75 %". The app's main interface has a light gray background. It features two text input fields: the first contains the URL "wss://demo.flashphoner.com:8443/" and the second is labeled "streamName". Below these fields is a toggle switch labeled "Publish system or mic audio", which is currently turned on (green). At the bottom of the visible interface is a large, rounded gray button with the text "Start Broadcast".

Application view when screen sharing is started



A special extension process is used to capture video from screen. This process works until device is locked or screen capturing is stopped manually.

Analyzing example code

To analyze the code take ScreenCapturer example version which is available on [GitHub](#).

Classes

- main application view class: ScreenCapturerViewController (implementation file [ScreenCapturerViewController.swift](#))
- extension implementation class: ScreenCapturerExtensionHandler (implementation file [ScreenCapturerExtensionHandler.swift](#))

1. Import API

code

```
import FPWCSPi2Swift
```

2. Screen capturer extension parameters setup

[code](#)

UserDefaults.suiteName parameter must be equal to extension application group id

```
@IBAction func broadcastBtnPressed(_ sender: Any) {
    ...
    pickerView.showsMicrophoneButton = systemOrMicSwitch.isOn

    let userDefaults = UserDefaults.init(suiteName: "group.com.flashphoner.ScreenCapturerSwift")
    userDefaults?.set(urlField.text, forKey: "wcsUrl")
    userDefaults?.set(publishVideoName.text, forKey: "streamName")
    userDefaults?.set(systemOrMicSwitch.isOn, forKey: "useMic")
    ...
}
```

3. Receiving screen capture parameters in extension code

[code](#)

```
override func broadcastStarted(withSetupInfo setupInfo: [String : NSObject]?) {
    ...
    let userDefaults = UserDefaults.init(suiteName: "group.com.flashphoner.ScreenCapturerSwift")
    let wcsUrl = userDefaults?.string(forKey: "wcsUrl")
    if wcsUrl != self.wcsUrl || session?.getStatus() != .fpwcsSessionStatusEstablished {
        session?.disconnect()
        session = nil
    }
    self.wcsUrl = wcsUrl ?? self.wcsUrl

    let streamName = userDefaults?.string(forKey: "streamName")
    self.streamName = streamName ?? self.streamName
    ...
}
```

4. Screen capturer object setup to capture audio

FPWCSPi2.getAudioManager().useAudioModule[code](#)

```
let useMic = userDefaults?.bool(forKey: "useMic")

capturer = ScreenRTCVideoCapturer(useMic: useMic ?? true)

FPWCSPi2.getAudioManager().useAudioModule(true)
```

5. Session creation to publish screen stream

WCSSession,WCSSession.connect [code](#)

```
if (session == nil) {
    let options = FPWCSPi2SessionOptions()
    options.urlServer = self.wcsUrl
    options.appKey = "defaultApp"
    do {
        try session = WCSSession(options)
    } catch {
        print(error)
    }
    ...
    session?.connect()
}
```

6. Screen stream publishing

WCSSession.createStream, WCSSession.publish [code](#)

The following parameters are passed to createStream method:

- stream name to publish
- ScreenRTCVideocapturer object to capture video from screen

```
func onConnected(_ session:WCSSession) throws {
    let options = FPWCSEApi2StreamOptions()
    options.name = streamName
    options.constraints = FPWCSEApi2MediaConstraints(audio: false, videoCapturer: capturer);

    try publishStream = session.createStream(options)
    ...
    try publishStream?.publish()
}
```

7. ScreenRTCVideocapturer class initialization

[code](#)

```
fileprivate class ScreenRTCVideocapturer: RTCVideoCapturer {
    let kNanosecondsPerSecond = 1000000000
    var useMic: Bool = true;

    init(useMic: Bool) {
        super.init()
        self.useMic = useMic
    }
    ...
}
```

8. System audio capturing in extension code

FPWCSEApi2.getAudioManager().getAudioModule().deliverRecordedData() [code](#)

```
func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with sampleBufferType: RPSampleBufferType) {
    switch sampleBufferType {
        ...
        case RPSampleBufferType.audioApp:
            if (!useMic) {
                FPWCSEApi2.getAudioManager().getAudioModule().deliverRecordedData(sampleBuffer)
            }
            break
        ...
    }
}
```

9. Microphone audio capturing in extension code

FPWCSEApi2.getAudioManager().getAudioModule().deliverRecordedData() [code](#)

```
func processSampleBuffer(_ sampleBuffer: CMSampleBuffer, with sampleBufferType: RPSampleBufferType) {
    switch sampleBufferType {
        ...
        case RPSampleBufferType.audioMic:
            if (useMic) {
                FPWCSEApi2.getAudioManager().getAudioModule().deliverRecordedData(sampleBuffer)
            }
            break
        ...
    }
}
```

Known limits

1. Music from iTunes will not play when system audio capturing is active.

2. ScreenCapturerSwift extension will receive a silence in `sampleBuffer` both from microphone and system audio if some other application uses the microphone. When microphone is released by other application, it is necessary to stop screen publishing and start it again to receive any audio.