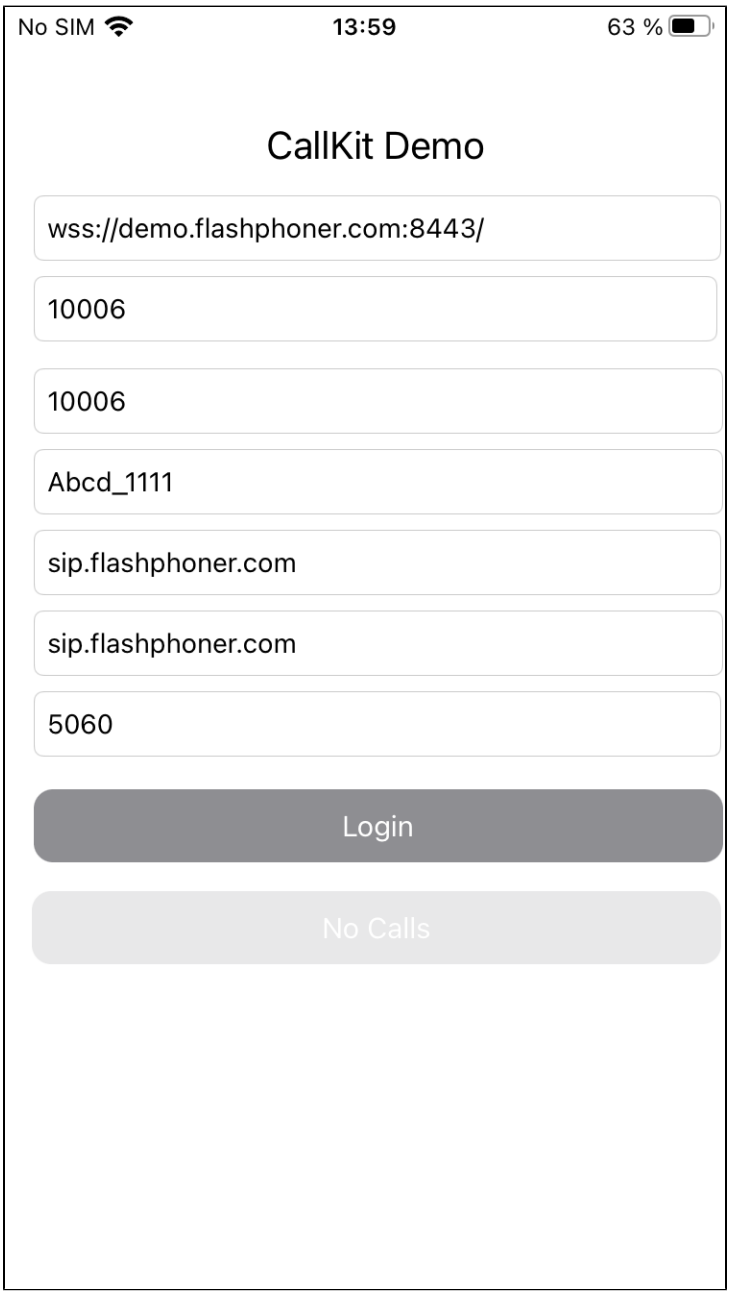
iOS Call Kit Demo Swift

- iOS application example using Call Kit to receive incoming SIP calls
- Server configuration
- Analyzing the code

iOS application example using Call Kit to receive incoming SIP calls

The example shows how to use Call Kit and push notifications to receive incoming SIP calls on iOS device.

Screenshot below contains input fields for SIP session credentials



The application receives push notification about an incoming SIP call even if it is in backgound or is closed. If the application is closed, it starts, connects to the SIP session on WCS server using session token received in notification, then accepts the incoming call.

Server configuration

The following parameters should be set on WCS server for push notifications to work

Parameter	Description
notification_apns_key_path	Apple Push Notification service key full name
notification_apns_key_id	APNs key Id
notification_apns_team_id	Apple developers team Id

For example

```
notification_apns_key_path=/opt/apns_auth_key.p8
notification_apns_team_id=SXZF5547NK
notification_apns_key_id=7NQA96WTFZ
```

WCS server sends notification to APNs according to these settings when incoming SIP call is received

Analyzing the code

To analyze the code, let's take CallKitDemo Swift example, which can be downloaded from GitHub

Aplication classes:

- class for the main view of the application: CallKitDemoViewController (file CallKitDemoViewController.swift)
- class for CXProviderDelegate protocol implementation: ProviderDelegate (file ProviderDelegate.swift)
- class to work with UI and push registry: AppDelegate (file AppDelegate.swift)
- extension to create CXAnswerCallAction object by user activity, to receive the call: NSUserActivity: StartCallConvertible (file NSUserActivity+StartCallConvertible.swift)
- extension to create CXAnswerCallAction object from application URL, to receive the call: URL: StartCallConvertible (file URL+StartCallConvertible.swift)
- extension to start call implementing INStartCallIntentHandling protocol (file CallKitIntentExtension/IntentHandler.swift)

1. Import of API

code

```
import FPWCSApi2Swift
```

2. Connection establishing to WCS server and SIP session creation

FPWCSApi2.createSession code

The following parameters are passed:

- urlServer - WCS server URL
- keepAlive - keep the SIP session alive when client disconnects
- sipRegisterRequired - register the SIP session on SIP PBX
- sipLogin - SIP account user login
- sipAuthenticationName - SIP accountuser authentication name
- sipPassword - SIP account password
- sipDomain - SIP PBX address
- sipOutboundProxy - SIP outbound proxy address (usually the same as SIP PBX address)
- sipPort - SIP PBX port
- noticationToken - push notification token
- appId - iOS application Id
- appKey - REST hook application key on WCS server

```
        let options = FPWCSApi2SessionOptions()
        options.urlServer = wcsUrl.text

        options.keepAlive = true

        options.sipRegisterRequired = true
        options.sipLogin = sipLogin.text
        options.sipAuthenticationName = sipAuthName.text
        options.sipPassword = sipPassword.text
        options.sipDomain = sipDomain.text
        options.sipOutboundProxy = sipOutboundProxy.text
        options.sipPort = Int(sipPort.text ?? "5060") as NSNumber?

        let userDefaults = UserDefaults.standard
        options.noticationToken = userDefaults.string(forKey: "voipToken")
        options.appId = "com.flashphoner.ios.CallKitDemoSwift"

        options.appKey = "defaultApp"

        do {
            let session = try FPWCSApi2.createSession(options)

            processSession(session)

            appDelegate.providerDelegate?.setSession(session)
            session.connect()
        } catch {
            print(error)
        }
```

3. Receiving the event about session successful creation

fpwcsSessionStatusEstablished code

Session connection data should be stored to reconnect when notification is received

```
    session.on(kFPWCSSessionStatus.fpwcsSessionStatusEstablished, callback: { rSession in
        NSLog("Session established")
        self.saveFields(rSession?.getAuthToken())
        self.toLogoutState()
    })
```

4. Saving WCSSession object and setting up session incoming call handlers

fpwcsCallStatusFinish, fpwcsCallStatusEstablished code

```
    func setSession(_ session: FPWCSApi2Session) {
        self.session = session;


        session.onIncomingCallCallback({ rCall in

            guard let call = rCall else {
                return
            }

            call.on(kFPWCSCallStatus.fpwcsCallStatusFinish, callback: {rCall in
                self.viewController.toNoCallState()

                guard let uuid = rCall?.getUuid() else {
                    return
                }
                self.provider.reportCall(with: uuid, endedAt: Date(), reason: .remoteEnded)
            })
            let id = call.getId()

            NSLog("CKD - session.onIncomingCallCallback. wcsCallId: " + (id ?? ""))


            call.on(kFPWCSCallStatus.fpwcsCallStatusEstablished, callback: {rCall in
                self.viewController.toHangupState(call.getId())
            })

            self.viewController.toAnswerState(call.getId())
            self.currentCall = call
            self.actionCall?.fulfill()
        })
    }
```

5. Describing answer call action request

code

```
    func answer(_ callId: String) {
        guard let call = self.session?.getCall(callId) else {
            return
        }
        let callController = CXCallController()
        let answerCallAction = CXAnswerCallAction(call: call.getUuid())
        callController.request(CXTransaction(action: answerCallAction),
                                            completion: { error in
                                                if let error = error {
                                                    print("Error: \(error)")
                                                } else {
                                                    print("Success")
                                                }
                                            })
    }
```

6. Performing answer call action

code

```
    func provider(_ provider: CXProvider, perform action: CXAnswerCallAction) {
        NSLog("CKD - CXAnswerCallAction: " + action.callUUID.uuidString)

        guard let call = self.session?.getCallBy(action.callUUID) else {
            if (self.session?.getStatus() == kFPWCSSessionStatus.fpwcsSessionStatusDisconnected || self.
session?.getStatus() == kFPWCSSessionStatus.fpwcsSessionStatusFailed) {
                self.session?.connect()
            }
            self.actionCall = action
            return
        }
        self.currentCall = call
        action.fulfill(withDateConnected: NSDate.now)
    }
```

7. Answering the call

call.answer code

```
    func provider(_ provider: CXProvider, didActivate audioSession: AVAudioSession) {
        NSLog("CKD - didActivate \(#function)")
        currentCall?.answer()
    }
```

8. Performing end call action

code

```
    func provider(_ provider: CXProvider, perform action: CXEndCallAction) {
        NSLog("CKD - CXEndCallAction: " + action.callUUID.uuidString)
        guard let call = session?.getCallBy(action.callUUID) else {
            action.fulfill()
            return
        }
        self.hangup(call.getId())
        action.fulfill()

    }
```

9. Hangup the call

call.hangup code

```
    func hangup(_ callId: String) {
        guard let call = self.session?.getCall(callId) else {
            return
        }
        call.hangup()
        self.provider.reportCall(with: call.getUuid(), endedAt: Date(), reason: .remoteEnded)
    }
```

10. Setting up token to receive notification

code

```
    func pushRegistry(_ registry: PKPushRegistry, didUpdate credentials: PKPushCredentials, for type:
PKPushType) {
        if (type == .voIP) {
            let token = credentials.token.map { String(format: "%02.2hhx", $0) }.joined()
            NSLog("CKD - Voip token: " + token)
            UserDefaults.standard.set(token, forKey: "voipToken")
        }
    }
```

11. Receiving push notification

code

```
    func pushRegistry(_ registry: PKPushRegistry, didReceiveIncomingPushWith payload: PKPushPayload, for type:
PKPushType) {
        guard type == .voIP else { return }
        if let id = payload.dictionaryPayload["id"] as? String,
           let uuidString = payload.dictionaryPayload["uuid"] as? String,
           let uuid = UUID(uuidString: uuidString),
           let handle = payload.dictionaryPayload["handle"] as? String
        {
            NSLog("CKD - pushRegistry uuidString: " + uuidString + "; id: " + id + "; handle: " + handle)
            providerDelegate?.reportIncomingCall(uuid: uuid, handle: handle, completion: nil)
        }
    }
```

12. Start audio call handler for intent extension

code

```
    func handle(intent: INStartAudioCallIntent, completion: @escaping (INStartAudioCallIntentResponse) -> Void)
{
        let response: INStartAudioCallIntentResponse
        defer {
            completion(response)
        }

        // Ensure there is a person handle
        guard intent.contacts?.first?.personHandle != nil else {
            response = INStartAudioCallIntentResponse(code: .failure, userActivity: nil)
            return
        }

        let userActivity = NSUserActivity(activityType: String(describing: INStartAudioCallIntent.self))

        response = INStartAudioCallIntentResponse(code: .continueInApp, userActivity: userActivity)
    }
```