

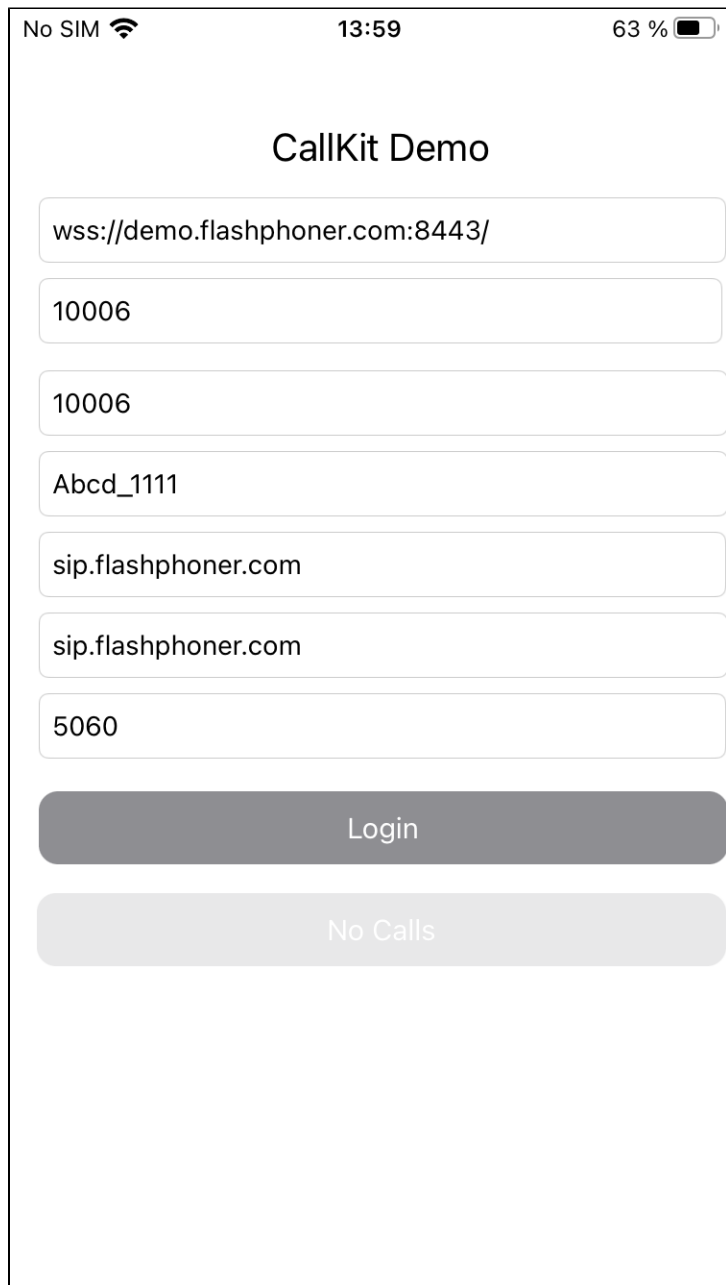
# iOS Call Kit Demo Swift

- [Пример iOS приложения для приема входящих звонков с использованием Call Kit](#)
- [Настройки сервера](#)
- [Работа с кодом примера](#)

## Пример iOS приложения для приема входящих звонков с использованием Call Kit

Данный пример может использоваться как основа для написания собственного приложения, использующего [Call Kit](#) и пуш-уведомлений для соединения с сервером и приема входящего SIP звонка.

На скриншоте приведены поля для ввода SIP параметров, необходимых для регистрации сессии



No SIM 13:59 63 %

CallKit Demo

wss://demo.flashphoner.com:8443/

10006

10006

Abcd\_1111

sip.flashphoner.com

sip.flashphoner.com

5060

Login

No Calls

При входящем звонке приложение получает пуш-уведомление, даже если оно свернуто или закрыто. Если приложение закрыто, оно подключается к SIP-сессии на сервере с токеном, полученным в уведомлении, и принимает входящий звонок.

# Настройки сервера

Для того, чтобы работали пуш-уведомления, на сервере необходимо указать следующие настройки

| Параметр                   | Описание                                                 |
|----------------------------|----------------------------------------------------------|
| notification_apns_key_path | Расположение файла ключа Apple Push Notification service |
| notification_apns_key_id   | Идентификатор ключа APNs                                 |
| notification_apns_team_id  | Идентификатор команды разработки                         |

Например

```
notification_apns_key_path=/opt/apns_auth_key.p8
notification_apns_team_id=5XZF5547NK
notification_apns_key_id=7NQA96WTFZ
```

В соответствии с этими настройками, сервер отправляет уведомления в APNs при поступлении входящего звонка

## Работа с кодом примера

Для разбора кода возьмем версию примера CallKitDemo Swift, которая доступна для скачивания на [GitHub](#)

Классы приложения:

- класс основного вида приложения CallKitDemoViewController (файл [CallKitDemoViewController.swift](#))
- класс реализации протокола CXProviderDelegate (файл [ProviderDelegate.swift](#))
- класс для работы с интерфейсом пользователя и реестром пуш-уведомлений AppDelegate (файл [AppDelegate.swift](#))
- расширение для создания объекта CXAnswerCallAction по действию пользователя, для ответа на звонок NSUserActivity: StartCallConvertible (файл [NSUserActivity+StartCallConvertible.swift](#))
- расширение для создания объекта CXAnswerCallAction из URL, для ответа на звонок URL: StartCallConvertible (файл [URL+StartCallConvertible.swift](#))
- расширение для создания звонка, реализующее протокол INStartCallIntentHandling (файл [CallKitIntentExtension/IntentHandler.swift](#))

### 1. Импорт API

[code](#)

```
import FPWCSPi2Swift
```

### 2. Подключение к серверу и создание SIP-сессии

FPWCSPi2.createSession [code](#)

При создании сессии передаются следующие параметры:

- urlServer - URL WCS сервера
- keepAlive - сохранять сессию при отключении клиента
- sipRegisterRequired - регистрировать сессию на SIP сервере
- sipLogin - имя пользователя на SIP сервере
- sipAuthenticationName - имя пользователя для аутентификации на SIP сервере
- sipPassword - пароль на SIP сервере
- sipDomain - адрес SIP сервера
- sipOutboundProxy - адрес SIP сервера
- sipPort - порт на SIP сервере
- notificationToken - токен для получения уведомлений о входящих звонках
- appId - идентификатор приложения
- appKey - REST hook приложение на WCS сервере

```

let options = FPWCSEApi2SessionOptions()
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.notificationToken = userDefaults.string(forKey: "voipToken")
options.appId = "com.flashphoner.ios.CallKitDemoSwift"

options.appKey = "defaultApp"

do {
    let session = try FPWCSEApi2.createSession(options)

    processSession(session)

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

```

### 3. Получение события об успешном создании сессии

fpwcsSessionStatusEstablished [code](#)

При этом сохраняются данные для подключения к сессии при получении уведомления о входящем звонке

```

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

```

### 4. Сохранение текущего объекта WCSession и настройка обработчиков входящего звонка для сессии

fpwcsCallStatusFinish, fpwcsCallStatusEstablished [code](#)

```

func setSession(_ session: FPWCSSession) {
    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. Запрос действия для ответа на звонок

[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. Выполнение действия для ответа на звонок

[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. Ответ на звонок

call.answer [code](#)

```

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

```

#### 8. Выполнение действия для завершения звонка

[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. Завершение звонка

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. Настройка данных для приема пуш-уведомлений

[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. Получение пуш-уведомлений

[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. Обработчик расширения для приема звонков

[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)
}
```