



Exotel Voice Client IOS SDK Integration Guide

1. Introduction	3
2. Licensing	3
3. Glossary	3
4. IOS SDK Integration	4
4.1. Getting Started	4
4.1.1. Software Package	4
4.1.2. Add Exotel Voice SDK Library to Project	4
4.1.3. Target Platform	4
4.1.4. Permissions	5
4.1.5. Proguard Rules	5
4.1.6. IOS Service	5
4.2. Initialize Library	5
4.3. Managing Calls	7
4.3.1. Make Outgoing Call	8
4.3.2. Receive Incoming Call	
4.3.3. Call Kit Integration	
4.3.4. Push Kit Integration	8
4.4. Audio Management	10
4.5. Call Statistics	10
4.6. Call Details	10
4.7. Handling of PSTN calls	10
4.8. Reporting Problems	11
4.9. Reporting Call Quality Feedback	11
5. Platform Integration	12
5.1. Customer API Endpoints	12
5.1.1. Dial Whom Endpoint	12
5.1.2. Push Notification Endpoint	13
5.2. Exotel API Endpoints	13
5.2.1. Subscriber Management	13
6. Authentication and Authorization	14
7. Reference Documents	14
8. Support Contact	15
9. Troubleshooting Guide	15
9.1. Outgoing Call	15

9.2. Incoming Call	15
9.3. Authentication	16

1. Introduction


ExotelVoice library enables you to add the voip calling feature into your app. This document outlines the integration steps. The library supports only peer to peer 2-way calls. Multi-party conferencing use cases are not supported.

[Section 4](#) describes integration steps for ExotelVoice IOS SDK. [Section 5.1](#) describes the webhooks that should be supported in your application backend for number-masking workflow. [Section 5.2.1](#) describes subscribers provisioning in the Exotel platform.

2. Licensing

Create a trial [account](#) in Exotel. To enable voip calling in the trial account, contact [exotel/support](#). Once Exotel support enables the voip capability in the account , a VOIP Exophone will be created as shown below and available in the account under the Exophones section .

ExoPhone

EXOPHONE	TYPE
095-138-86363  Pin: 7022-1678-17	Not set
080-471-12327	Landline
sip-:08-040408080	VoIP

SIP Exophones discussed later in the section refers to Exophones of Voip type as shown above.

NOTE :- SIP and VOIP are used interchangeably in the document

3. Glossary


Terminology	Description
App	Mobile application
Application Backend	Customer backend service for the application
Client	User / Subscriber / Client signing up to use the mobile app
Customer	Exotel's customer licensing the SDK
Voip	Voice over IP

4. IOS SDK Integration

4.1. Getting Started

4.1.1. Software Package

The *ExotelVoice* SDK software package includes

- IOS [ExotelVoice.framework.zip](#) file of the SDK
- Integration Guide
- Sample application source code for reference: [exotel-voice-sample.zip](#)
- Doc for SDK API reference: [exotel-ios-voice-sample-doc.zip](#)
- Subscriber Management API Documentation:  Subscriber Management API

4.1.2. Add Exotel Voice SDK Library to Project

Configure `ExotelVoice.Framework`

- unzip `ExotelVoice.framework.zip`
- Copy `ExotelVoice.framework` folder under your project root directory

4.1.3. Target Platform

- Supported IOS Version: iOS 10+
- Supported IOS Sdk Arch : arm64

4.1.4. Permissions

- Record permission
- Notification permission

4.1.5. Proguard Rules - Not applicable

4.1.6. IOS Service

The application needs to integrate *ExotelVoice* in a service. This should be run as a foreground service when a call is in progress so that the application is not killed when it moves to the background. When the call is over, service should be moved to the background. Refer to *VoiceAppService* in the sample application.

4.2. Initialize Library

`ExotelVoice` Interface Class provides an entry point to initialize the voice library and set it up for handling calls.

Exotel Voice Client iOS SDK Integration Guide

```
// Get Exotel Voice Client
exotelVoiceClient = ExotelVoiceClientSDK.getExotelVoiceClient()

//set the event listener for sdk logs.
exotelVoiceClient?.setEventListener(eventListener: self)

// Set listener to handle events from voice client
extension VoiceAppService: ExotelVoiceClientEventListener {

    // Initialization success handler
    public func onInitializationSuccess() { ... }

    // Initialization failure handler
    public func onInitializationFailure(error: ExotelVoiceError) {...}

    /// Indicates de-initialization of the SDK
    public func onDeinitialized(){...}

    // Logs reported by the voice library
    public func onLog(level: LogLevel, tag: String, message: String) {...}

    // Log upload success handler
    public func onUploadLogSuccess() {...}

    // Log upload failure handler
    public func onUploadLogFailure(error: ExotelVoiceError) {...}

    // Authentication failure handler
    public func onAuthenticationFailure(error: ExotelVoiceError) {...}
}
```

SDK initialization requires a *subscriberToken* generated by the Exotel platform. Workflow for this token generation and management is described in section [Authentication and Authorization](#).

```
// Request token from application backend (example)
String subscriberToken = getAccessToken();
```

The *subscriberToken* is provided to *ExotelVoice* during initialization.

```
// Initialize client library
let content = ["DeviceId": UIDevice.current.identifierForVendor?.uuidString,
              "DeviceType": "ios"]

exotelVoiceClient?.initialize(
    context: content as [String : Any], // iOS context
    hostname: hostname,                // Exotel Voice Platform Host URL
    subscriberName: subscriberName,    // To generate token for subscriber
    displayName: displayName,          // Display name of the subscriber
    accountSid: accountSid,            // Exotel Account SID
    subscriberToken: subscriberToken   // Token fetched from Exotel platform
)
```

Client library notifies *onInitializationSuccess()* on success and *onInitializationFailure()* on failure. On expiry of *subscriberToken*, the library will post *onAuthenticationFailure()* at which application should request new *subscriberToken* from application backend and program in the *initialize()* API.

Parameter	Description
Hostname	https://miles.apac-sg.exotel.in/v2
SubscriberName	Param "subscriber_name" returned as part of the Subscriber management API to create a subscriber.
DisplayName	Subscriber name.
AccountSid	Account SID param from API settings page in the Exotel Dashboard.
SubscriberToken	can be gotten from the API in the 'Get Subscriber Token' section in the Subscriber Management API document . In the <i>subscriberToken</i> , both the <i>refresh token</i> and <i>access token</i> are base64 encoded.

Subscriber token format example

```
"subscriber_token":
{
  "refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOiJlNzY2NDc5OTksImV4cCI6MTU3Njc0Nzk5OSwiY2xpZW50X2lkIjoibWUwNTg2NUUifQ.Hc3umVfFlKIPIj8R9kcP9o9hE9he51le08rO22u7eqs",
  "access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOiJlNzY2NDc5OTksImV4cCI6MTU3Njc0Nzk5OSwiY2xpZW50X2lkIjoibWUwNTg2NUUifQ.Hc3umVfFlKIPIj8R9kcP9o9hE9he51le08rO22u7eqs"
}
```

Below are the sample methods to get the subscriberToken, Customers can implement this in their own method.

Use the below HTTP APIs to get the subscriber token and convert subscriber token to object and pass it to `exotelVoiceClient.initialize()`;

Note:

- Copying manually generated subscriber token has issues. Best way is to make API calls in source code and pass them by converting to jobject to the `exotelVoiceClient.initialize()`;
- Pass the **proper device ID** to the login api to avoid invalid refresh token error.
- Device_id is the actual device id in which the app is running using below sample code.

```
"device_id": UIDevice.current.identifierForVendor?.uuidString
```

Generate access token:

Refer “Subscriber Management API” documentation section 3.1 for creating subscriber token.

4.2.1. Stopping SDK

To de-initialize the sdk, `ExotelVoiceClient` exposes stop api to de-initialize the SDK.

```
exotelVoiceClient?.stop();
```

when sdk gets de-initialized,
client SDK will send callback `onDeinitialized()`

4.3. Managing Calls

After successful initialization, the library is ready to handle calls. Dialing-out calls or receiving incoming calls is managed through *CallController* Interface.

```
// Get Call Controller interface  
callController = self.exotelVoiceClient?.getCallController()
```

Call progress events are notified to the application through the *CallListener* Interface attached to the *CallController* object.

```
// Attach call listener  
extension VoiceAppService: CallListener {  
  
    // Handler for incoming call alert  
    public func onIncomingCall(call: Call) { . . . }  
  
    // Handler for outgoing call initiated event  
    public func onCallInitiated(call: Call) { . . . }  
  
    // Handler for call ringing event for outgoing call  
    public func onCallRinging(call: Call) { . . . }  
  
    // Handler for call connected event  
    public func onCallEstablished(call: Call) { . . . }
```

```
// Handler for call termination event
public func onCallEnded(call: Call) { . . . }

// Handler for missed call alert
public func onMissedCall(remoteId: String, time: Date) { . . . }

}
```

Each call object provides a *Call* Interface to manage the call lifecycle and perform operations like answer incoming calls, hangup calls, mute, unmute, and get call statistics.

```
// Mute
mCall?.mute()

// Terminate call
mCall?.hangup()
```

4.3.1. Make Outgoing Call

To make outgoing calls, the *CallController* interface must be created and a listener interface is attached as mentioned in section [Managing Calls](#). Provide sip exophone number in *dial()* API of *CallController* Interface to make outgoing calls.

```
// Get Call Controller interface
self.callController = self.exotelVoiceClient?.getCallController()

// Attache call progress listener
self.callController?.setCallListener(callListener: self)

// Dial out call to remoteId
call = callController?.dial(remoteId: <value>, message: <value>)
```

Params for dial method -

Param	Mandatory	Description
remoteId	Yes	<p>ExotelVoice library always routes the call via SIP Exophone configured for your account. SIP Exophones are different from the PSTN / Mobile Exophones. They can be identified using sip: prefix. Contact exotel support to enable SIP Exophone in your account.</p> <p>Similar to the number masking workflow, the application backend must maintain the call context between caller and callee. On receiving the call at SIP Exophone, the exotel platform will request the callee details from the application</p>

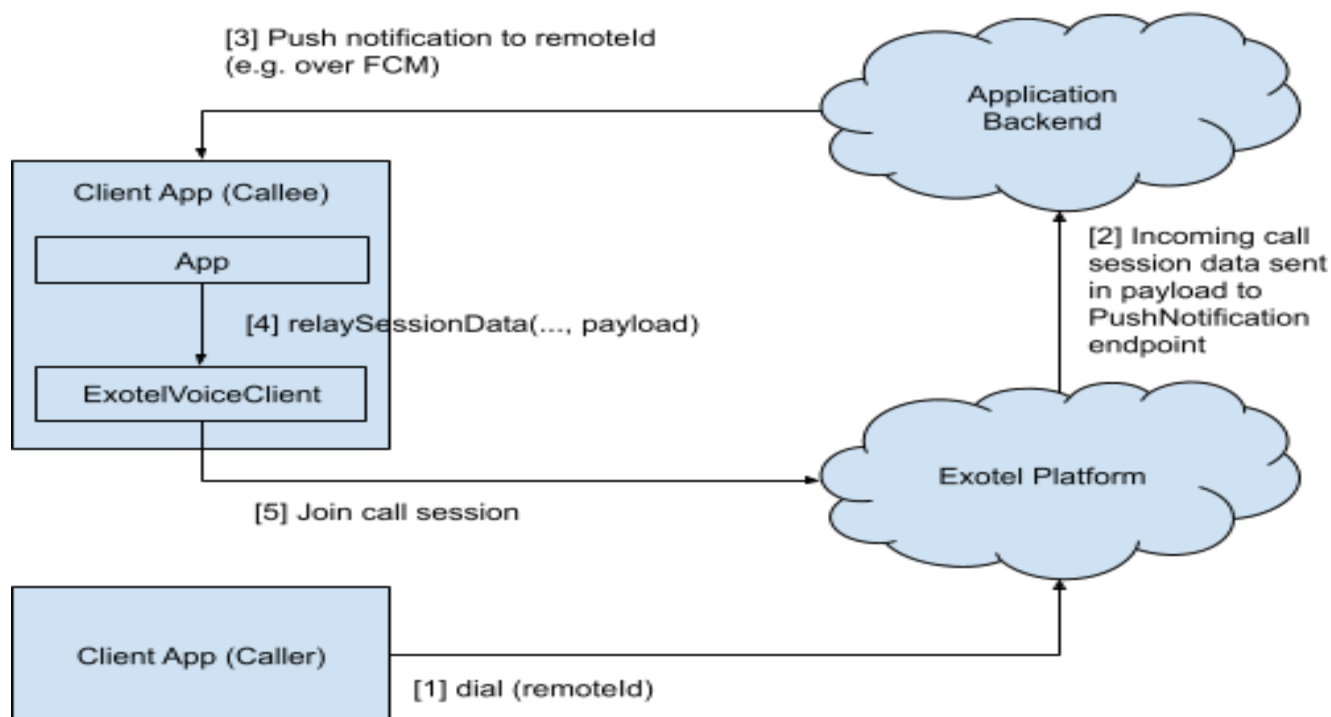
		backend to bridge the call. Refer section Dial Whom Endpoint for details.
message	No	The string that will be passed here will be sent to the DialWhom Endpoint under the “CustomField” param. All alphanumeric characters are allowed along with comma `,` colon `:` and all kinds of brackets - <code>{ } () []</code>

Once initiated, call progress events can be monitored through *CallListener* Interface and call control operations can be performed using *Call* Interface. After dialing, the application can use *hangup()* API on the call object to cancel / terminate outgoing calls.

```
// Cancel / Terminate call
mCall?.hangup()
```

4.3.2. Receive Incoming Call

The Mobile application receives incoming call data in Push Notification sent by your application backend. Refer section [Push Notification Endpoint](#) for details.



To handle the incoming call, the *ExotelVoice* library should be in an initialized state. Refer to section [Initialize Library](#). After initialization, the *CallController* Interface should be created and the listener be attached as mentioned in section [Managing Calls](#).

Exotel Voice Client iOS SDK Integration Guide

```
// Get Call Controller interface
self.callController = self.exotelVoiceClient?.getCallController()

// Attache call progress listener
self.callController?.setCallListener(callListener: self)
```

The sample application uses Firebase Cloud Messaging to push call data to the device. Once received at the mobile app, it is sent to *ExotleVoiceClient* through *relaySessionData* API.

```
extension AppDelegate: UNUserNotificationCenterDelegate {
func application(_ application: UIApplication,
                 didReceiveRemoteNotification userInfo: [AnyHashable:
Any],
                 fetchCompletionHandler completionHandler: @escaping
(UIBackgroundFetchResult)
                 -> Void) {

// decode json and build session data structure
sessionData["payload"] = payload
sessionData["payloadVersion"] = payloadVersion
sessionData["subscriberName"] = userId

exotelVoiceClient?.relaySessionData(payload: sessionData)

}
```

The voice library will process and validate the data and send an *onIncomingCall()* event to the application with the call object. The application can call the *answer()* API to accept the incoming call.

```
// Accept call
mCall?.answer()
```

The application can call *hangup()* API on the call object to decline an incoming call.

```
// Decline / Terminate call
mCall?.hangup()
```

4.3.3. Call Kit Integration

If your app is integrated with Call Kit, refer to the following [lifecycle diagram](#) to ensure there are no conflicts.

User Interaction:

- On the left side, we see a user icon representing interactions with a sample app.
- The user initiates actions like “click dial” for outgoing calls and “accept call” or “reject call” for incoming ones.

Pre-requisites :

- **CallKit UUID** will manage by your App.

```
private static var activeUUID: UUID?
```

- **CXProviderDelegate** is subscriber by App to handle the call kit actions

```
let providerConfiguration = CXProviderConfiguration(localizedName: "Exotel Sample App")
providerConfiguration.maximumCallsPerCallGroup = 1
providerConfiguration.supportsVideo = false
providerConfiguration.supportedHandleTypes = [.phoneNumber]
providerConfiguration.ringtoneSound = "Ringtone.aif"
if let iconImage = UIImage(named: "AppIcon") {
    providerConfiguration.iconTemplateImageData = iconImage.pngData()
}

provider = CXProvider(configuration: providerConfiguration)
callController = CXCallController()
provideDelagete = ProviderDelegate(provider: provider!)
provider!.setDelegate(provideDelagete, queue: nil)
```

Here **ProviderDelegate** is custom class which has implemented the **CXProviderDelegate**.

Outgoing Calls:

1. Start outgoing calls by calling dial() api of exotel sdk.

```
callController?.dial(remoteID: destination, message: message)
```

2. Then after make sure to call the start Transaction with **CXStartCallAction** (using the UUID).

```
let handle = CXHandle(type: .phoneNumber, value: destination)
activeUUID = UUID()
let startCallAction = CXStartCallAction(call: activeUUID!, handle: handle)
startCallAction.isVideo = video

let transaction = CXTransaction()
transaction.addAction(startCallAction)

requestTransaction(transaction)
```

3. Call-kit UI will start in background
4. Call kit delegate method for action **CXStartCallAction** will get executed

```
func provider(_ provider: CXProvider, perform action: CXStartCallAction) {  
    setActiveUUID(uuid: action.callUUID)  
    action.fulfill()  
}
```

5. When sdk send **onCallRinging()** callback to app, app will start call kit timer for ringing after updating the status to ringing state.

```
provider?.reportOutgoingCall(with: activeUUID!, startedConnectingAt: nil)
```

6. Timer will start in call kit UI.

Incoming Calls:

1. When relay session data (push notification data) triggers an incoming call notification, the CallKit UI should be set up to display the incoming call with options to accept or reject. Users should be provided with the option to accept the call. They can choose to accept the call using the function **reportIncomingCall**.

```
class func reportIncomingCall(uuid: UUID, handle: String, hasVideo: Bool =  
false, completion: ((Error?) -> Void)? = nil) {  
    // Construct a CXCallUpdate describing the incoming call, including  
    the caller.  
    let update = CXCallUpdate()  
    update.remoteHandle = CXHandle(type: .phoneNumber, value: handle)  
    update.hasVideo = hasVideo  
  
    // Report the incoming call to the system  
    provider!.reportNewIncomingCall(with: uuid, update: update) { error in  
        /*  
        Only add an incoming call to an app's list of calls if it's  
        allowed, i.e., there is no error.  
        Calls may be denied for various legitimate reasons. See  
        CXErrorCodeIncomingCallError.  
        */  
        if let error = error {  
            VoiceAppLogger.error(TAG: CallKitUtils.TAG, message: "Error  
requesting transaction: \(error)")  
        } else {  
            VoiceAppLogger.info(TAG: CallKitUtils.TAG, message: "Requested  
transaction successfully:")  
        }  
    }  
}
```

- Upon choosing to accept the call, the system should invoke the ***CXAnswerCallAction*** to answer the call.

```
func provider(_ provider: CXProvider, perform action: CXAnswerCallAction) {
    VoiceAppLogger.info(TAG: TAG, message: "CXAnswerCallAction")
    VoiceAppService.shared.answer()
    // Signal to the system that the action was successfully performed.
    action.fulfill()
}
```

- Alternatively, users should also be provided with the option to reject the call. They can choose to reject the call using the ***CXEndCallAction***.

```
func provider(_ provider: CXProvider, perform action: CXEndCallAction) {
    VoiceAppLogger.info(TAG: TAG, message: "CXEndCallAction")
    do {
        try VoiceAppService.shared.hangup()
    } catch let error {
        VoiceAppLogger.debug(TAG: TAG, message: "Error:
\\(error.localizedDescription)")
    }
    // Signal to the system that the action was successfully performed.
    action.fulfill()
}
```

Connected Calls:

- Once a call is established, its status is updated in the UI.
- The timer is started in the call kit once the call is connected.

Ending Calls:

- Both the sample app and the CallKit UI can initiate the action.
- When the user ends the call, the sample app starts a transaction with ***CXEndCallAction*** (using the UUID).
- Function *hangup()* is called.

```
public func hangup() throws {
    if (nil == mCall) {
        let message = "Call Object is NULL"
        VoiceAppLogger.error(TAG: TAG, message: message)
        throw VoiceAppError(module: TAG, localizedDescription: message)
    }
    do {
        try mCall?.hangup()
    } catch {
        VoiceAppLogger.error(TAG: TAG, message: "Exception in call hangup with
CallId: \\(String(describing: mCall?.getCallDetails().getCallId()))")
    }
}
```

```
}
```

The home view updates after ending a call.

4.3.4. PushKit Integration

PushKit is used to manage VoIP call notifications and handle incoming call events with CallKit.

- **Initializing PushKit**

The PushKit registry is initialized inside the `AppDelegate` class within the `didFinishLaunchingWithOptions` method.

```
var pushRegistry: PKPushRegistry = PKPushRegistry(queue: DispatchQueue.main)
pushRegistry.delegate = self
pushRegistry.desiredPushTypes = [.voIP]
```

- **Implementing `PKPushRegistryDelegate`**

An extension of the `AppDelegate` class conforms to the `PKPushRegistryDelegate` protocol to handle PushKit-related events.

```
extension AppDelegate: PKPushRegistryDelegate {
    func pushRegistry(_ registry: PKPushRegistry, didUpdate pushCredentials:
        PKPushCredentials, for type: PKPushType) {
        guard type == .voIP else { return }

    }

    func pushRegistry(_ registry: PKPushRegistry, didReceiveIncomingPushWith
        payload: PKPushPayload, for type: PKPushType, completion: @escaping () ->
        Void) {
        completion()
    }
}
```

- **Receiving the PushKit Token**

The `pushRegistry(_:didUpdate:for:)` method is triggered upon app launch or whenever the PushKit token changes. This token is unique for the device and must be sent to the backend.

```
extension AppDelegate: PKPushRegistryDelegate {
```

```
func pushRegistry(_ registry: PKPushRegistry, didUpdate pushCredentials:
PKPushCredentials, for type: PKPushType) {
    guard type == .voIP else { return }

    let token = pushCredentials.token.map { String(format: "%02x", $0)
}.joined()
    if token.isEmpty {
        VoiceAppLogger.debug(TAG: TAG, message: "PushKit Token is not
generated yet!!")
        return
    }

    VoiceAppLogger.debug(TAG: TAG, message: "PushKit token: \(token)")
    UserDefaults.standard.set(token, forKey:
UserDefaults.Keys.firebaseToken.rawValue)
}
}
```

- **Handling Incoming Push Notifications**

The ***pushRegistry(_:didReceiveIncomingPushWith:for:completion:)*** method is triggered when the app receives a PushKit notification. It processes the payload and triggers the CallKit UI to display the incoming call screen.

```
func pushRegistry(_ registry: PKPushRegistry, didReceiveIncomingPushWith
payload: PKPushPayload, for type: PKPushType, completion: @escaping () ->
Void) {
    VoiceAppLogger.debug(TAG: TAG, message: "VoIP push received")

    ApplicationUtils.checkMicrophonePermission { isEnabled in
        guard isEnabled else {
            UserDefaults.standard.set("false", forKey:
UserDefaults.Keys.isLoggedIn.rawValue)
            completion()
            return
        }
    }

    ApplicationUtils.checkNotificationsPermission { isEnabled in
        guard isEnabled else {
            UserDefaults.standard.set("false", forKey:
UserDefaults.Keys.isLoggedIn.rawValue)
            completion()
            return
        }
    }

    guard let payloadDict = payload.dictionaryPayload as? [String: Any],
        let callerId = payloadDict["subscriberName"] as? String else {
        VoiceAppLogger.error(TAG: TAG, message: "Invalid payload format or
missing callerId")
        completion()
        return
    }

    VoiceAppService.shared.pushNotificationData = payloadDict
}
```

```

    let validateLogin = UserDefaults.standard.string(forKey:
UserDefaults.Keys.isLoggedIn.rawValue) ?? "false"
    guard validateLogin != "false" else {
        VoiceAppLogger.error(TAG: TAG, message: "User is not logged into App")
        completion()
        return
    }

    CallKitUtils.displayIncomingCall(handle: callerId)
    completion()
}

```

- **Answering the Call**

When the user accepts the call via the CallKit UI by tapping the accept button, the ***provider(_:perform:CXAnswerCallAction)*** method in the ***CXProvider*** delegate is triggered. This method initializes the audio session and call the ***sendPushNotificationData()*** method.

```

func provider(_ provider: CXProvider, perform action: CXAnswerCallAction) {
    VoiceAppLogger.info(TAG: TAG, message: "CXAnswerCallAction")

    guard let payload = VoiceAppService.shared.pushNotificationData,
        let userId = payload["subscriberName"] as? String,
        let payloadVersion = payload["payloadVersion"] as? String,
        let payloadData = payload["payload"] as? String else {
        VoiceAppLogger.error(TAG: TAG, message: "Missing data in payload")
        return
    }

    VoiceAppLogger.debug(TAG: TAG, message: "Handling the notification on
answer button tap")
    self.startAudioSessionIfNeeded()

    DispatchQueue.main.asyncAfter(deadline: .now()) {
        VoiceAppService.shared.sendPushNotificationData(
            payload: payloadData,
            payloadVersion: payloadVersion,
            userId: userId
        )
    }

    action.fulfill()
}

```

- **Activating the Audio Session**

Calling the ***startAudioSessionIfNeeded()*** method initializes and activates the ***AVAudioSession*** to ensure it is correctly configured for voice call functionality. It uses the ***AVAudioSession*** framework to manage audio behaviors..

```

func startAudioSessionIfNeeded() {
    do {
        let session = AVAudioSession.sharedInstance()
    }
}

```



```

        try session.setCategory(.playAndRecord, mode: .voiceChat, options:
[.duckOthers, .allowBluetooth, .defaultToSpeaker])
        try session.setActive(true)
    } catch {
        VoiceAppLogger.error(TAG: TAG, message: "Error starting audio session:
\\(error)")
    }
}

```

- **Incoming Call Management**

After receiving an incoming call notification and displaying the CallKit UI, the app handles the call by playing a ringtone and invoking the **answer()** method to accept the call. This is managed within the **onIncomingCall** function.

```

public func onIncomingCall(call: Call) {
    VoiceAppLogger.debug(TAG: TAG, message: "Incoming Call Received, CallId:
\\(call.getCallDetails().getCallId()) remoteId:
\\(call.getCallDetails().getRemoteId())")

    tonePlayback.playRingTone()
    mCall = call

    DispatchQueue.main.asyncAfter(deadline: .now() + 1) {
        self.answer()
    }
}

```

4.4. Audio Management

ExotelVoice can detect the change in audio output selection. Default audio output mode is the earpiece. When a wired headset is plugged in or a Bluetooth headset is paired with a phone, it automatically routes audio over a new route. Speaker phone mode can be enabled using *Call* object APIs.

```

// Enable speaker mode
mCall?.enableSpeaker()

// Disable speaker mode
mCall?.disableSpeaker()

```

Local mic can be muted and unmuted during in-call using *Call* object APIs.

```
// Enable speaker mode
mCall?.mute()

// Disable speaker mode
mCall?.unmute()
```

Bluetooth mode can be enabled and disabled using *Call* object APIs. This will only work when any bluetooth device is connected to the phone.

```
/// Enables bluetooth mode for the call
mCall?.enableBluetooth()

/// Disables bluetooth mode for the call
mCall?.disableBluetooth()
```

4.5. Call State

The Call State can be accessed by calling *getLatestCallDetails().getCallState()*. This state can be displayed on the calling screen to let the user know about the state of the call. Refer to the [Call State Flow diagram](#).

Call State	Description	Call Back	Recommended message to display on the calling screen
NONE	This state occurs when a call object is created but not yet initiated.	-	-
OUTGOING_INITIATED	This state is set when an outgoing call is initiated. It's set after dialing and before the call starts ringing.	-	Connecting
EARLY	Indicates early media reception before the call is answered.	-	-
RINGING	The receiver's phone is ringing but hasn't been answered yet.	onCallRinging()	Ringing

CONNECTING	The call starts connecting.	-	Connecting
INCOMING	This state is set when there's an incoming call, and the system isn't idle.	onIncomingCall()	Caller ID, custom information related to the callee
ANSWERING	Occurs when the user accepts the incoming call; it's a transitional phase before the call gets established.	-	Answering
ESTABLISHED	Indicates that both parties have accepted the call, and communication is established.	onCallEstablished()	Connected
MEDIA_DISRUPTED	Occurs if there's a disruption in media transmission during an ongoing call due to issues like RTP loss. To handle such disruptions effectively ensuring continuity or termination based on severity it initiates reconnection procedures and moves to RECONNECTING if RTP loss persists. To ensure calls remain active during temporary disruptions it monitors RTP resumption or port renewal activities and returns to ESTABLISHED upon successful media restoration.	onMediaDisrupted()	Reconnecting

RENEWING_MEDIA	This state indicates that media (RTP) is resuming or renewing after being disrupted.	onRenewingMedia()	Reconnecting
ENDING	The process of ending or dropping an ongoing or established call starts here	-	-

4.6. Call Statistics

Application can query the call quality statistics periodically during the call by using *getStatistics()* API of the *Call* object. It provides info about codec and the call QoS parameters like packet loss, jitter and round trip delay.

4.7. Call Details

Application can query the call details record of the call using *getCallDetails()* API of the *Call* interface. The CDR provides info on callId, call state, call duration, call end reason etc. This API can be queried only for the latest call since SDK maintains only a single call context.

4.8. Handling of PSTN calls

Exotel/Voice has following behavior during PSTN calls:

- When a PSTN call is in progress, no outgoing calls are allowed by the voice client.
- When a PSTN call is in progress, any incoming VoIP call is automatically declined by the voice library and reported as a missed call to the application.
- When a VoIP call is in progress and a PSTN call lands on the phone, then the VoIP call continues if the user declines or does not respond to the PSTN call. If the user accepts the PSTN call, then the VoIP call is automatically terminated by the voice client.

4.9. Handling of PSTN calls when Call Kit is Integrated

Exotel/Voice has following behavior during PSTN calls:

- When a PSTN call is in progress, no outgoing calls are allowed by the voice client.
- The following scenarios will be handled by the Call Kit instead of the SDK -
 - When a PSTN call is in progress and a VoIP call is incoming
 - When a VoIP call is in progress and a PSTN call lands on the phone

For these scenarios, Call kit provides options to the end user to either reject and accept the incoming call or hold and accept the incoming call.

4.10. Reporting Problems

The voice client library logs are stored in the application internal storage. Any issue along with the logs can be reported by app to Exotel using *uploadLogs()* API of *Exotel/Voice* Interface.

```
// Upload logs with description from startDate and endDate
exotelVoiceClient?.uploadLogs(startDate: startDate, endDate: endDate,
description: description)
```

The API triggers *onUploadLogSuccess()* or *onUploadLogFailure()* callback based on the success or failure of the operation.

4.11. Reporting Call Quality Feedback

Call quality can be queried from the user and reported to the exotel platform at the end of the call.

```
// Upload logs with description from startDate and endDate
int rating = 3
CallIssue issue = BACKGROUND_NOISE
mPreviousCall?.postFeedback(rating: rating, issue: issue)
```

The rating is a quality score that can take values 1 to 5.

```
5 - Excellent quality. No issues.
4 - Good quality. Negligible issues.
3 - Average quality. Minor audio noise.
2 - Bad quality. Frequent choppy audio or high audio delay.
1 - Terrible. Unable to communicate. Call drop, No-audio or one-way audio.
```

The call issue is a descriptive explanation of the issue.

NO_ISSUE	No issues observed
BACKGROUND_NOISE	Low audio clarity due to noisy audio
CHOPPY_AUDIO	Frequent breaks in audio or garbling in audio
HIGH_LATENCY	Significant delay in audio
NO_AUDIO	No audio received from far user
ECHO	Echo during the call

5. Platform Integration

5.1. Customer API Endpoints

Exotel provides full control to customers to implement call routing business logic. For this, customers need to host the following HTTP endpoints to handle callbacks from Exotel platform.

5.1.1. Dial Whom Endpoint

Host a HTTP endpoint which will be queried by the Exotel platform to get to the destination user.

Method: GET

Request URL (Example): <https://company.com/v1/accounts/exotel/dialtonumber>

Note: This URL needs to be provided to Exotel or configured in the connect applet.

Request Body:

- CallSid - unique call identifier
- CallFrom - caller username
- CallTo - exophone
- CustomField - It will be passed only if you have sent the second optional param while making the call from the app.

Expected Response:

On success, the API should return with response code 200 OK. The response body should be a string of type *sip:<remoteId>*. "sip:" tag is needed to hint the exotel platform to connect calls over voip. At present, SIP and PSTN intermixing is not supported. Any other response is treated as failure. remoteId is the username with which the call destination subscriber was registered with Exotel platform.

Example:

Request

```
GET
/v1/accounts/exotelip2ipcalling1/dialtonumber?CallSid=743ddcf5a0552050bc37b6d0
ff9613bn&CallFrom=sip:Alice&CallTo=sip:08040408080&CallStatus=ringing&Directio
n=incoming&Created=Sat,+23+Nov+2019+22:00:37&DialCallDuration=0&StartTime=2019
-11-23+22:00:37&EndTime=1970-01-01+05:30:00&CallType=call-attempt&DialWhomNumb
er=&flow_id=249196&tenant_id=113828&From=sip:Alice&To=sip:08040408080&CurrentT
ime=2019-11-23+22:00:37
```

Response

```
{
  "fetch_after_attempt": false,
  "destination": {
    "numbers": [
      "sip:1234567890"
    ]
  },
  "outgoing_phone_number": "08080808080",
  "record": false,
  "recording_channels": "dual",
  "max_ringing_duration": 30,
  "max_conversation_duration": 3600,
  "request_id": "2e48100e6b474714b1a64bfa9f5b7a55",
  "method": "GET",
  "http_code": 200,
  "code": null,
```

```

    "error_data": null,
    "status": null
  }

```

5.1.2. Push Notification Endpoint

Exotel implements registration-less dialing where a user need not periodically register with SIP registrar for receiving incoming calls. Instead the call details are pushed to the device as push notification using which call can be established. This method is beneficial as calls will reach the user even when the app is not in foreground or swipe killed. It saves battery since it does not need to keep persistent voip connection when idle.

Exotel will provide call data to this endpoint that needs to be pushed to the client device from your application backend.

Note - Headers are not supported in the notify endpoints.

Method: POST

Request URL (Example):

`https://<your_api_key>:<your_api_token>@company.com/v1/accounts/exotel/pushtoclient`

Note: This URL needs to be configured in Exotel platform

Request Body:

- subscriberName - Name with which subscriber registered with Exotel platform
- payload - call data which should be passed to the client SDK
- payloadVersion - version of payload scheme

Expected Response:

On success, the API should return with response code 200 OK. Any other response is treated as failure.

5.2. Exotel API Endpoints

5.2.1. Subscriber Management

Customers can manage their subscribers in the Exotel platform using the APIs listed “*Subscriber-Management-API*” document.

For clients to be able to use voip calling features they need to be added as subscribers under your account. There are two ways in which your client registration with Exotel account happens,

a. Pre-provisioning

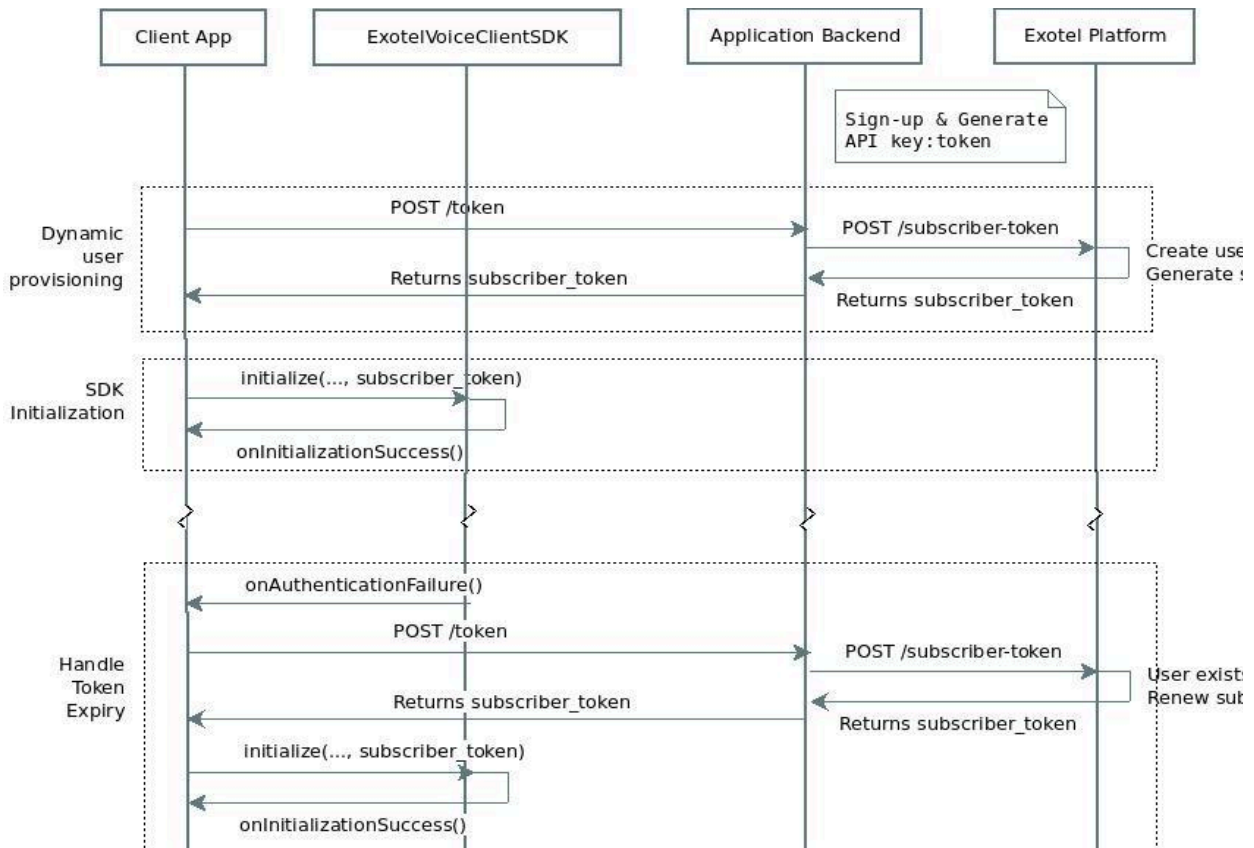
The customer pre-configures the client accounts even before the app is installed by the client. Refer to the “*Subscriber-Management-API*” document for details on creating client accounts.

b. Dynamic provisioning

A client account is created after the app is installed by the user and signs-up with the application backend. Refer to [Authentication and Authorization](#) for workflow details. Once client provisioning happens, clients can be managed using [Subscriber Management APIs](#).

6. Authentication and Authorization

Access to the Exotel platform by *ExotelVoice* is authenticated using a set of bearer tokens - *subscriber_token*. The Application backend must obtain these tokens from the Exotel platform and provide them to the client on request. Refer to the “*Subscriber-Management-API*” document for details.



On receiving the `onAuthenticationFailure` event, the application should request a new *subscriber_token* from its backend and reinitialize the SDK as shown in section [Initialize Library](#). Contact [exotel support](#) if you get `onAuthenticationFailure` even after token renewal.

`onAuthenticationError()` event provides following error types:

- `AUTHENTICATION_INVALID_TOKEN` - token parameters are invalid
- `AUTHENTICATION_EXPIRED_TOKEN` - token expired

7. Reference Documents

- IOS JavaDoc for the *ExotelVoice* library API
- *Subscriber-Management-API* document for details on API to manage subscriber

8. Support Contact

Please write to hello@exotel.in for any support required with integration.

9. Troubleshooting Guide

9.1. Outgoing Call

- a. I am not getting any requests on the “Dial Whom” endpoint for outgoing calls?

- i. Check if “Dial Whom” URL was configured in [Connect Applet](#) and reachable.
- ii. Check if the call is listed in your account dashboard. If not, then
 - Check if the phone has internet connectivity.
 - Check if the SIP Exophone number was set in *CallController::dial()* API.
 - Check if *CallListener::onCallFailed()* event was received with error type CONGESTION_ERROR, which means rate limit quota was exceeded.
 - Check if *CallListener::onCallInitiated()* event was received from the SDK.
 - Check if *Call::getCallDetails()* returns non-null *sessionId* field
 - Check if
- iii. Report the issue to [exotel support](#) with *sessionId* (from App) and *Call-Reference-Id* (if available, from account dashboard)

- b. I am hearing a ringing tone but the callee has not received the call?

Ringing tone implies that the call has reached the exotel platform.

9.2. Incoming Call

- a. My calls are not landing on the callee app?

- i. Check if the call is listed in your account dashboard and dialout happened to the remote subscriber. If not then, refer to the troubleshooting guide for outgoing calls.
- ii. Check if the application received push notification with call payload from your application backend. If not, then
 - Check the device logs
 - Check if [Push Notification URL](#) is reachable and configured in the exotel platform.
 - Check if Push Notification URL got the call payload from exotel platform.
 - Check if the call payload was sent to the target device from your backend.

- iii. If the call notification is received in the app, then
 - Check if the library is initialized using *ExotelVoiceClient?.isInitialized()*
 - Check if payload received in push notification was programmed in the library through *ExotelVoiceClient?.relaySessionData()*
- iv. Report the issue to [exotel support](#) with *Call-Reference-Id* (from account dashboard)

9.3. Authentication

- a. Requesting subscriber-token from exotel platform giving 4xx response?

Response Code	Troubleshooting
400	Malformed packet. Check subscriber_name format. It should be an alphanumeric string of size less than 16 characters.
401	It requires basic authentication using API key:token
403	VoIP calling is not enabled in your account. Contact exotel support.

- b. Why is SDK reporting *onAuthenticationFailure* response for a subscriber token received from exotel platform?

i. Check the <i>ErrorType</i> in response. If it is,	
• AUTHENTICATION_INVALID_TOKEN	Invalid Token. Initialize new token
• AUTHENTICATION_EXPIRED_TOKEN	Token expired. Initialize new token
ii. Contact exotel support if the issue persists.	