



# Exotel Voice Client Android SDK Integration Guide

|  |           |
|--|-----------|
| <b>1. Introduction</b>                         | <b>1</b>  |
| <b>2. Licensing</b>                            | <b>1</b>  |
| <b>3. Glossary</b>                             | <b>2</b>  |
| <b>4. Android SDK Integration</b>              | <b>2</b>  |
| 4.1. Getting Started                           | 2         |
| 4.1.1. Software Package                        | 2         |
| 4.1.2. Add exotel Voice SDK Library to Project | 2         |
| 4.1.3. Target Platform                         | 3         |
| 4.1.4. Permissions                             | 3         |
| 4.1.5. Proguard Rules                          | 3         |
| 4.1.6. Android Service                         | 4         |
| 4.2. Initialize Library                        | 4         |
| 4.3. Managing Calls                            | 5         |
| 4.3.1. Make Outgoing Call                      | 6         |
| 4.3.2. Receive Incoming Call                   | 7         |
| 4.4. Audio Management                          | 8         |
| 4.5. Call Statistics                           | 9         |
| 4.6. Call Details                              | 9         |
| 4.7. Handling of PSTN calls                    | 9         |
| 4.8. Reporting Problems                        | 9         |
| 4.9. Reporting Call Quality Feedback           | 10        |
| <b>5. Platform Integration</b>                 | <b>10</b> |
| 5.1. Customer API Endpoints                    | 10        |
| 5.1.1. Dial Whom Endpoint                      | 10        |
| 5.1.2. Push Notification Endpoint              | 11        |
| 5.2. Exotel API Endpoints                      | 12        |
| 5.2.1. Subscriber Management                   | 12        |
| <b>6. Authentication and Authorization</b>     | <b>12</b> |
| <b>7. Reference Documents</b>                  | <b>13</b> |
| <b>8. Support Contact</b>                      | <b>13</b> |
| <b>9. Troubleshooting Guide</b>                | <b>13</b> |
| 9.1. Outgoing Call                             | 13        |
| 9.2. Incoming Call                             | 14        |

### 9.3. Authentication

15

## 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 Android 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     |
|--|----------|
| <a href="#">095-138-86363</a> <br>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. Android SDK Integration

### 4.1. Getting Started

#### 4.1.1. Software Package

The *ExotelVoice* SDK software package includes

- Android AAR file
- Integration Guide
- Sample application for reference
- JavaDoc for API reference
- Subscriber Management API Documentation

#### 4.1.2. Add Exotel Voice SDK Library to Project

**Method I:**

- Open your Android application code in Android Studio
- Choose File -> New -> New Module -> Import exotel-voice-release.aar Package
- Provide the path to the downloaded AAR file
- Add the SDK as a dependency for the app
  - a. Go to File -> Project structure -> Dependencies > "+" that says "All Dependencies > Add path of AAR"
  - b. Add Exotel Voice SDK as dependency to your application

**Method II:**

- Copy the AAR file to the libs folder and edit the build.gradle file to include

```
implementation project(path: ':exotel-voice-sdk')
```

- Below contents part of settings.gradle file

```
include ':app', ':exotel-voice-sdk'  
implementation fileTree(include: ['*.jar'], dir: 'libs')
```

- Add the following libraries to the build.gradle file

```
implementation 'com.google.code.gson:gson:[2.8.5] '
implementation 'dnsjava:dnsjava:[2.1.9,)'
implementation 'com.squareup.okhttp3:okhttp:[3.14.2,)'
implementation 'com.squareup.okhttp3:logging-interceptor:[3.14.2,)'
```

## 4.1.3. Target Platform

- Supported Android Version: 10.0+
- Supported Android ABIs: armeabi-v7a, arm64-v8a, x86\_64 and x86

## 4.1.4. Permissions

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.BROADCAST_STICKY" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.DISABLE_KEYGUARD" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />

//Required for Android 14 onwards
<uses-permission
android:name="android.permission.FOREGROUND_SERVICE_PHONE_CALL" />
<uses-permission
android:name="android.permission.FOREGROUND_SERVICE_MICROPHONE" />
```

## 4.1.5. Proguard Rules

The proguard rules for *ExotelVoice* are included in the AAR file. These rules will be auto included when the proguard is enabled for your application build.

## 4.1.6. Android 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.

### Use Foreground Service Types

When setting up the Foreground Service, define the `foregroundServiceType` attributes as `phoneCall` and `microphone` in the manifest. This will indicate that the service is responsible for handling calls and microphone access, preventing unnecessary security issues related to background access. For instance:

```
<service android:name=".VoiceAppService"
android:foregroundServiceType="phoneCall|microphone"
android:permission="android.permission.FOREGROUND_SERVICE"
```

```
android:exported="false" />
```

This setup ensures that the service continues running during an active call and has the necessary permissions to access the microphone. Additionally, after the call ends, the service can be moved to the background to release unnecessary system resources.

### Managing Microphone Permission According to Android 14 Guidelines

Android 14 enforces strict restrictions on background services, particularly for microphone access. To comply with these guidelines, apps must manage foreground services effectively, ensuring they are correctly configured for both incoming and outgoing calls. This ensures smooth call functionality while maintaining security and privacy.

When the foreground service is started, it should begin with `FOREGROUND_SERVICE_TYPE_PHONE_CALL`. This allows the app to operate in the background while displaying the incoming call notification.

```
ServiceCompat.startForeground(this, NOTIFICATION_ID,  
notification, ServiceInfo.FOREGROUND_SERVICE_TYPE_PHONE_CALL);
```

### Incoming Calls

When the app is closed and an incoming call is detected, it should first start as a normal service to handle initializing the SDK

Once the user got the `onIncomingCall` event from the SDK and trying to answer the call, so before invoking the answer method of the SDK, the application must be in foreground and the service type must be updated to enable microphone access during the call. This will ensure that app will work seamlessly when the app is running in the background or is closed.

,

```
public void answer() throws Exception {  
    VoiceAppLogger.debug(TAG, "Answering call");  
    if (null == mCall) {  
        String message = "Call object is NULL";  
        throw new Exception(message);  
    }  
    try {  
        updateForegroundServiceType(mCall, CallState.ANSWERING);  
        tonePlayback.stopTone();  
    }  
}
```

```
mCall.answer();
} catch (Exception e) {
    throw new Exception(e.getMessage());
}
VoiceAppLogger.debug(TAG, "After Answering call");
}
```

### Outgoing Calls

After dialing the call from the app the user will get the `onCallInitiated()` callback. Once the callback is received, the service should be updated to enable microphone access for the call.

Note that , while updating the foreground service with microphone , application must be in foreground. It will ensure that , if during call, app goes into background then voice will flow properly.

```
@Override
public void onCallInitiated(Call call) {
    VoiceAppLogger.debug(TAG, "on Call initiated");
    for (CallEvents callEvents : callEventListenerList) {
        callEvents.onCallInitiated(call);
    }
    mCall = call;
    updateForegroundServiceType(call, CallState.OUTGOING_INITIATED);
    VoiceAppLogger.debug(TAG, "End: onCallInitiated");
}
```

### Update Foreground Service Type Function

The `updateForegroundServiceType` method is responsible for updating the foreground service type based on the current call state. It ensures compliance with Android 14's guidelines for background services and microphone access.

```
private void updateForegroundServiceType(Call call, CallState
outgoingInitiated) {
    handler.post(new Runnable() {
        @Override
        public void run() {
            Notification notification =
utils.createNotification(outgoingInitiated,
call.getCallDetails().getRemoteId(), call.getCallDetails().getCallId(),
call.getCallDetails().getCallDirection());
            if (Build.VERSION.SDK_INT < Build.VERSION_CODES.TIRAMISU) {
                startForeground(NOTIFICATION_ID, notification);
            } else {
                startForeground(NOTIFICATION_ID, notification,
ServiceInfo.FOREGROUND_SERVICE_TYPE_MANIFEST);
            }
        }
    });
}
```

```

    });
}

```

## 4.2. Initialize Library

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

```

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

//set the event listener for sdk logs.
exotelVoiceClient.setEventListener(this);

// Set listener to handle events from voice client
exotelVoiceClient.setEventListener(new ExotelVoiceClientListener() {

    // Initialization success handler
    void onInitializationSuccess() { ... }

    // Initialization failure handler
    void onInitializationFailure(ExotelVoipError exotelVoipError) { ... }

    // Logs reported by the voice library
    void onLog(LogLevel logLevel, String Tag, String Message) { ... }

    // Log upload success handler
    void onUploadLogSuccess() { ... }

    // Log upload failure handler
    void onUploadLogFailure(ExotelVoipError exotelVoipError) { ... }

    // Authentication failure handler
    void onAuthenticationFailure(ExotelVoipError exotelVoipError) { ... }
});

```

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
android.content.Context context = this.getApplicationContext();
exotelVoiceClient.initialize(

```



## Exotel Voice Client Android SDK Integration Guide

```
context,          // Android application context
hostname,
subscriberName,
displayName,
accountSid,       // Exotel Account SID
subscriberToken,
);
```

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        | <a href="https://miles.apac-sg.exotel.in/v2">https://miles.apac-sg.exotel.in/v2</a>  |
| SubscriberName  | Param "subscriber_name" returned as part of the <a href="#">Subscriber management API</a> to create a subscriber.  |
| DisplayName     | Subscriber name.   |
| AccountSid      | <a href="#">Account SID param from API settings page</a> in the Exotel Dashboard.  |
| SubscriberToken | can be gotten from the API in the 'Get Subscriber Token' section in the <a href="#">Subscriber Management API document</a> . 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.eyJpc3MiOiJleG90ZWw",
  "access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWw"
}
```

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 jobject 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.
- build.gradle should be proper. Complete working build.gradle file is attached in the link <https://github.com/exotel/exotel-voip-sdk-android/blob/main/build.gradle>
- Device\_id is the actual device id in which the app is running using below sample code.

```
String androidId = Settings.Secure.getString(context.getContentResolver(),  
Settings.Secure.ANDROID_ID);
```

### Generate access token:

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

## 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 callController = exotelVoiceClient.getCallController();
```

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

```
// Attach call listener
callController.setCallListener(new CallListener() {

    // Handler for incoming call alert
    void onIncomingCall(Call call) { ... }

    // Handler for outgoing call initiated event
    void onCallInitiated(Call call) { ... }

    // Handler for call ringing event for outgoing call
    void onCallRinging(Call call) { ... }

    // Handler for call connected event
    void onCallEstablished(Call call) { ... }

    // Handler for call termination event
    void onCallEnded(Call call) { ... }

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

});
```

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
call.mute();

// Terminate call
call.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
CallController callController = exotelVoiceClient.getCallController();

// Attache call progress listener
callController.setCallListener(new CallListener() { ... }

// Dial out call to remoteId
Call call = callController.dial(<sip exophone>, <custom_field>);
```

Params for dial method -

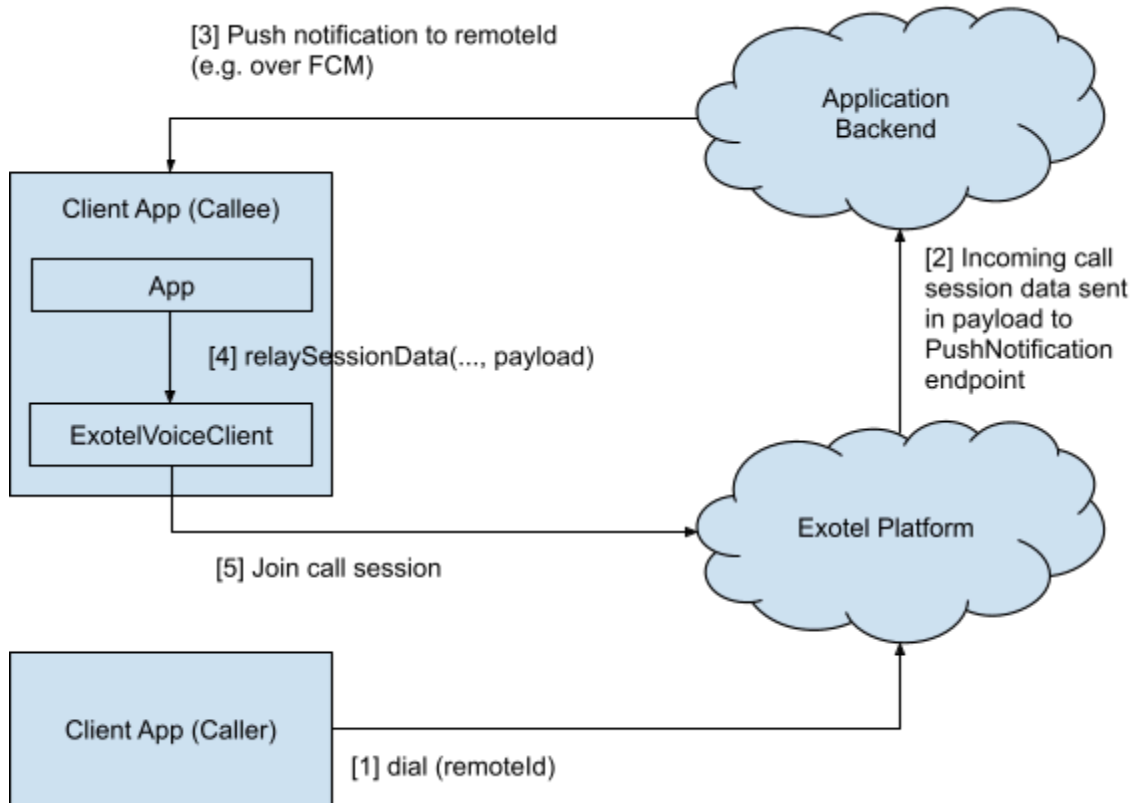
| Param | Mandatory | Description |
|-------|-----------|-------------|
|-------|-----------|-------------|

|              |     |   |
|--------------|-----|---|
| sip_exophone | 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 <a href="#">exotel support</a> 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 backend to bridge the call. Refer section <a href="#">Dial Whom Endpoint</a> for details.</p> |
| custom_field | No  | <p>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></p>   |

Once initiated, call progress events can be monitored through *CallListener* Interface and call control operations can be performed using *Call* Interface.

#### 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](#).

```
// Get Call Controller interface
CallController callController = exotelVoiceClient.getCallController();

// Attache call progress listener
callController.setCallListener(new CallListener() { ... })
```

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

```
public class VoiceAppFirebaseService extends FirebaseMessagingService{
    public void onMessageReceived(RemoteMessage remoteMessage) {

        Map<String,String> callData = remoteMessage.getData();
        Map<String,String> pushNotificationData = new HashMap<>();

        pushNotificationData.put("payload", callData.get("payload"));
        pushNotificationData.put(
            "payloadVersion", callData.get("payloadVersion"));

        exotelVoiceClient.relaySessionData(pushNotificationData);
    }
}
```

```
}
```

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

Application has to update their foreground service with microphone permission.

And it must be required to app in foreground mode to update the foreground service with microphone permission. Hence for this we can move application in foreground with UI element. And then update the foreground service.

The application can call the *answer()* API to accept the incoming call.

```
// Accept call
call.answer();
```

#### 4.3.3. Hangup Call

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

```
// Decline / Terminate call
call.hangup();
```

### 4.4. Audio Management

Before the call begins, Exotel's SDK checks the current audio output mode and sets the audio route accordingly. By default, it is in the earpiece mode. For example, If a wired headset is plugged in, audio will get routed via wired headset.

Speaker phone mode can be enabled and disabled using *Call* object APIs.

```
// change audio route to speaker
call.enableSpeaker();

// change audio route to phone earpiece
call.disableSpeaker();
```

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

```
// change audio route to bluetooth. Only works if a bluetooth device is
connected to the phone.
```

```
call.enableBluetooth();

// change audio route to phone earpiece
call.disableBluetooth();
```

Current Audio route can be fetched using Call object API `getAudioRoute`. It will return `CallAudioRoute` Enum i.e `EARPIECE`, `SPEAKER`, `BLUETOOTH`, `HEADPHONES`

```
// get current audio route
call.getAudioRoute();
```

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

```
// mute the call
call.mute();

// unmute the call
call.unmute();
```

## 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.   | onCallRingin()             | Ringin  |
| 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()<br>( ) | 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 | onMediaDisrupted()<br>( )  | Reconnecting  |



|                |  |                         |              |
|----------------|--|-------------------------|--------------|
|                | ESTABLISHED upon successful media restoration.                                       |                         |              |
| 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         | -                       | -            |
| ENDED          | Indicates that a call has ended completely, returning system status to idle.         | onCallEnded: ENDED      | -            |
| -              | This indicates that the media session has been cleaned up.                           | onDestroyMediaSession() | -            |

#### 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

*ExotelVoice* 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. 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 *ExotelVoice* Interface.

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

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

## 4.10. 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
call.postFeedback(rating, 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:<remoteld>*. "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. remoteld 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 as payload & payloadVersion to your push notification endpoint that needs to be pushed to the client device from your application backend.

Refer section [Receive Incoming Call](#) for handling of push notification payload.

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

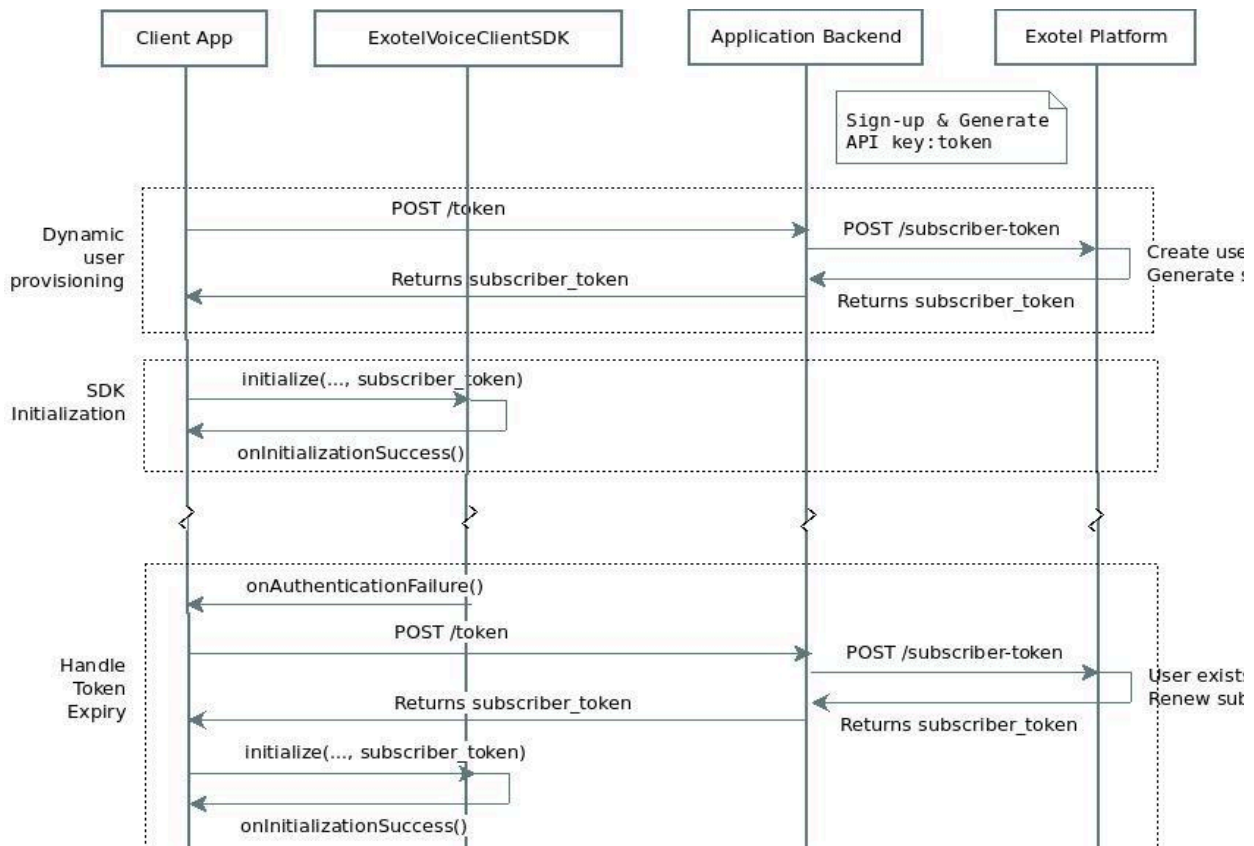
The customer pre-configures the client accounts even before the app is installed by the client. Refer to the “*Subscriber-Management-API*” document for details on creating client accounts.

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

- Android 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](mailto: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 OEM. If Xiaomi phones, refer to section [Problems with push notifications on Xiaomi devices](#).
  - 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)

b. Problems with push notifications on Xiaomi devices

- i. Your application needs to have the following additional permissions in the MIUI settings page\*. Open settings (by tapping the cog icon in the top right corner), select “Manage Apps”, select “your application”
  - Enable “Autostart”
  - Select “Other permissions”
    - Turn on “Show on Lock Screen”
    - Turn on “Display pop-up windows while running in the background”
  - Select “Battery Saver” and set “No restrictions”

\* The steps here refer to MIUI Global 11.0.2 on Android 7.1. Depending on device and software version, designation and location of menu items may differ.

c. Ringtone not playing while receiving push notifications on Xiaomi devices

Your application needs to have the following additional permissions in the MIUI settings page\*. Open settings (by tapping the cog icon in the top right corner), select “Manage Apps”, select “your application”. Go to “Notifications” → “Notification Categories”

- Enable “Allow floating notification” for ‘Exotel Voip Sample’
- Enable “Allow floating notification” for ‘io.cloudonix.sdk.ForegroundNotificationChannelName’

\* The steps here refer to MIUI Global 13.0.7.0 on Android 12.016. Depending on device and software version, designation and location of menu items may differ.

## d. Display Not Awake

Display awake is handled from the following permission defined in manifest:  
`<uses-permission android:name="android.permission.WAKE_LOCK" />`

It has been found that the ringtone is playing but the display is not awake, the above permission can't handle it properly. This can be handled from the application side using PowerManager implementation [Light up screen when notification received android](#) which is not handled in current app code.

### 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.<br>Initialize new token |
| • AUTHENTICATION_EXPIRED_TOKEN                                    | Token expired. Initialize new token    |
| ii. Contact <a href="#">exotel support</a> if the issue persists. |  |