exotel
DOCS

# Programmable Voice

# CONTENTS

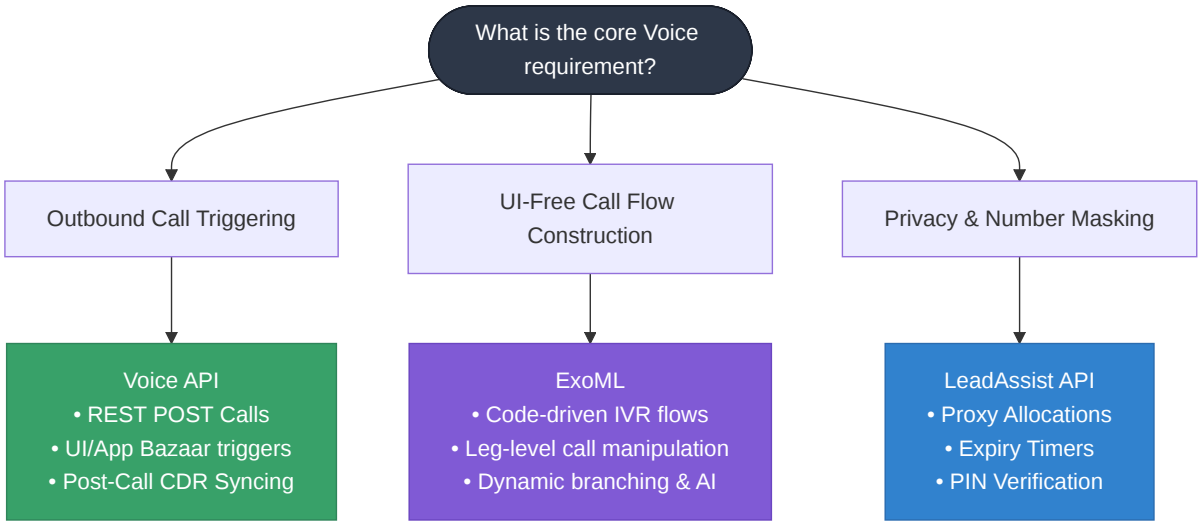
# 1. Developer Guide: Navigating Exotel's Programmable Voice APIs

Exotel offers three primary programmable Voice API families—**Voice APIs**, **ExoML**, and **LeadAssist**. Each suite caters to distinct execution patterns, control models, and UI dependencies. This guide helps developers select the right integration path based on architectural requirements, call flow complexity, and privacy needs.

## Architectural Overview

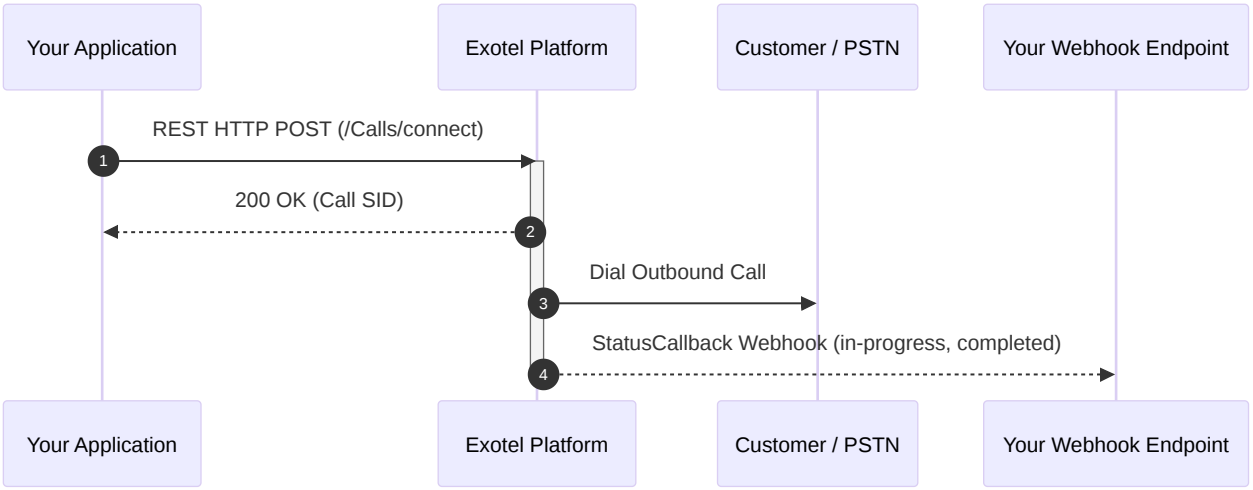
| API Suite | Control Model | Call Flow Definition | Core Purpose |
|---|---|---|---|
| **Voice APIs** | Server-Initiated (Push) | Dashboard / App Bazaar UI | Triggering outbound calls, pulling CDRs, simple click-to-call |
| **ExoML** | Event-Driven (Webhooks) | Pure Code (No UI Dependency) | Programmatic call flows, dynamic IVRs, leg-level manipulation |
| **LeadAssist** | Managed Proxy Allocation | Managed Rules | Privacy-safe 2-party proxy calls, virtual number masking |

## API Selection Decision Tree




## 1. Voice V1 APIs: REST-Driven Triggering & Reporting

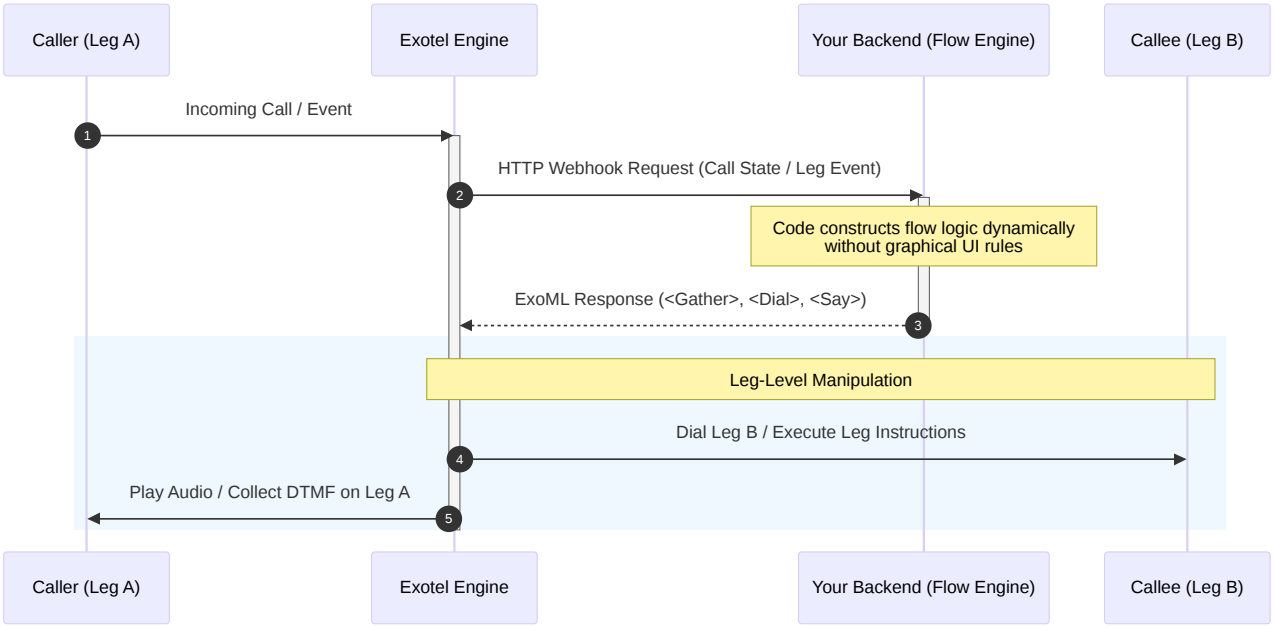
The Voice V1 API suite provides a RESTful interface to trigger outbound calls and extract historical call data without maintaining active state on your backend during the call.




- **Key Capabilities:**
  - **Click-to-Call:** Programmatically initiates a connection between two phone numbers using an ExoPhone virtual number.
  - **Flow Triggering:** Triggers outbound calls that drop recipients into pre-configured graphical IVR flows created in Exotel's App Bazaar UI.
  - **Reporting & Recordings:** Synchronously fetches Call Detail Records (CDRs), recording URLs, and telecom circle metadata.
- **When to Use:** Outbound transactional calls, CRM click-to-call buttons, and post-call analytics ingestion.
- **When NOT to Use:** When call routing needs to be constructed and altered dynamically in code during an active call.

## 2. ExoML APIs: Programmatic Call Flow Engine

ExoML (Exotel Markup Language) eliminates reliance on dashboard-based visual IVR builders. It allows developers to construct, version-control, and execute entire call flows directly in code via webhooks and XML verbs.

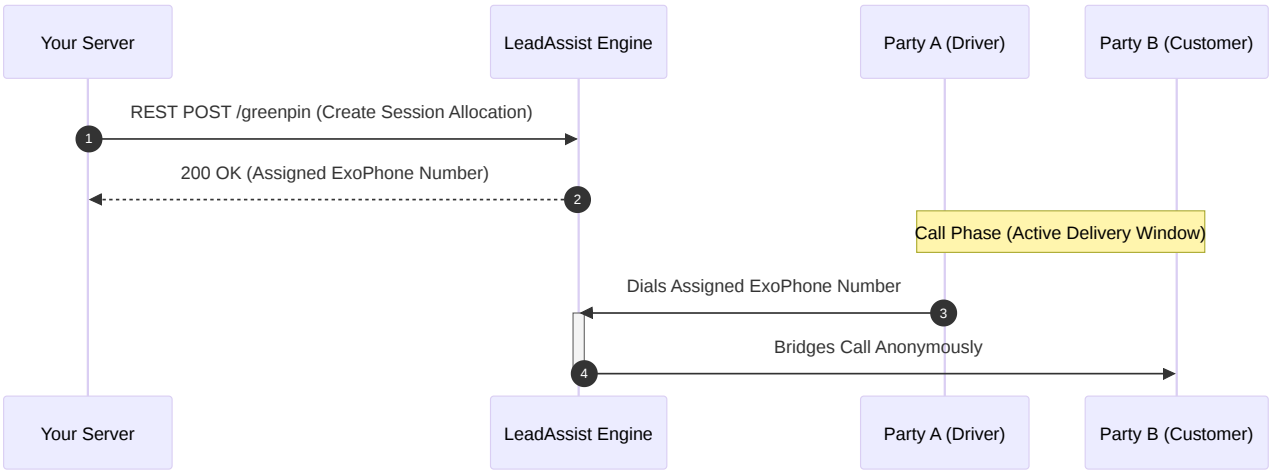



- **Key Capabilities:**
  - **UI-Free Flow Construction:** Build complex multi-stage IVRs completely within your application logic and Git repositories.

  - **Leg-Level Manipulation:** Target instructions to specific call legs asynchronously—placing individual callers on hold, executing targeted audio prompts, or performing warm agent transfers.
  - **Dynamic Runtime Branching:** Route calls conditionally based on live database lookups, customer state, or dynamic DTMF keypresses.
  - **Real-time Media & AI Streaming:** Pipe audio to external WebSocket streams for live transcription or generative AI voice assistant integration.
- **When to Use:** Dynamic IVRs built inside code, applications requiring granular leg-level orchestration, and AI voicebot streaming.
- **When NOT to Use:** Basic outbound triggers where a fixed visual dashboard flow or standard 2-party connection is sufficient.

## 3. LeadAssist APIs: Privacy Proxy & Masked Communication

LeadAssist is a domain-specific API engine designed to manage masked communications between two parties without exposing personal phone numbers.




- **Key Capabilities:**
  - **Dual-Sided Masking:** Hides phone numbers for both caller and recipient using a shared virtual ExoPhone.
  - **Dynamic Expiry:** Sets custom lifespans on number mappings (e.g., active only for a 30-minute delivery window).
  - **GreenPin Verification:** Restricts connection access via optional dynamic PIN verification prompts.
- **When to Use:** Rideshare apps, last-mile delivery, and two-sided buyer/seller marketplaces.
- **When NOT to Use:** Internal contact centers, general IVRs, or standard business telephony calls.

# 2. Voice APIs

## 2.1. Connect Voice AI API

The Connect to Bot API lets you programmatically call a phone number and connect the call to a conversational AI bot in real time. Audio flows in both directions – the caller speaks to the bot, and the bot responds to the caller – over a secure WebSocket connection.

**Typical use cases:** AI-powered outbound campaigns, virtual customer care agents, automated surveys, and appointment reminders with live confirmation.

### Before You Start

You will need:

- Your Exotel Account SID, API Key, and API Token (available from your Exotel Dashboard).
- An ExoPhone (virtual number) to use as the Caller ID.
- A WebSocket server endpoint (your bot service) that accepts audio in real time.

### How It Works

1. You send an API request with the caller's phone number and your bot's WebSocket URL.
2. Exotel dials the phone number.
3. Once the call is answered, Exotel opens a connection to your WebSocket server.
4. Audio is streamed live between the caller and your bot for the duration of the call.
5. When the call ends (by either party), Exotel sends a status update to your callback URL.

### API Reference

**Endpoint:**

```
POST /v1/Accounts/{AccountSid}/Calls/connect
```

**Base URLs:**

| Region | Base URL |
|---|---|
| Singapore | https://api.exotel.com |
| Mumbai | https://api.in.exotel.com |

Authentication uses HTTP Basic Auth with your API Key as the username and API Token as the password.

#### Parameters

Required

| Parameter | Description |
|---|---|
| From | The phone number to call. Use international format (e.g., +919876543210). |
| CallerId | Your Exotel virtual number that appears as the caller ID. |
| StreamUrl | The WebSocket URL of your bot server. Must start with ws:// or wss://. |
| StreamType | Set this to bidirectional to enable two-way audio. |

Optional

| Parameter | Description |
|---|---|
| Record | Set to true to record the call. Default: false. |
| RecordingChannels | single (merged) or dual (separate track per side). Default: single. |
| TimeLimit | Maximum call duration in seconds. Up to 14,400 (4 hours). |
| CustomField | Any text you want attached to the call record (max 128 characters). Useful for tracking order IDs, customer IDs, etc. |
| StatusCallback | A URL on your server that Exotel will call with updates when the call status changes. |
| StatusCallbackEvents | Which events to receive: answered, terminal (call ended), or ringing. You can specify more than one. This is available only for leg1 |
| StreamName | An optional label for the stream, up to 32 characters. |

## Example Request

bash

```
curl -X POST 'https://<api_key>:<api_token>@api.exotel.com/v1/Accounts/<AccountSid>/Calls/connect' \  -F 'StreamType=bidirectional' \  -F 'StreamUrl=wss://your-bot.example.com/media' \  -F 'From=+91XXXXXXXXXX' \  -F 'CallerId=0XXXXXXXXXX' \  -F 'Record=true' \  -F 'StatusCallback=https://your-server.com/callback' \  -F 'StatusCallbackEvents[]=terminal'
```

## Response

A successful request returns a Call object immediately. The call is still being set up at this point – use StatusCallback to track when the call is answered and when it ends.

json

```
{  "Call": {    "Sid": "a1b2c3d4e5f6...",    "Status": "in-progress",    "From": "+91XXXXXXXXXX",  "PhoneNumberSid": "0XXXXXXXXXX",    "Direction": "outbound-api",    "DateCreated": "2025-06-01 10:00:00",  "RecordingUrl": null  }}
```

| Field | What it means |
|---|---|
| Sid | A unique ID for this call. Save it to look up call details later. |
| Status | Current state of the call. |
| From | The number that was dialed. |
| RecordingUrl | Link to the recording once the call is complete (if recording was enabled). |

**Call status values:**

| Status | Meaning |
|---|---|
| queued | Call is being prepared. |
| in-progress | Call is active. |
| completed | Call ended normally. |
| failed | Call could not be placed. |
| busy | The number was busy. |
| no-answer | The number did not answer. |

## Configuring Your Bot's WebSocket Server

Your bot must accept a WebSocket connection and handle audio in real time. Key points:

- Exotel sends audio as raw PCM or mulaw at 8 kHz by default. To request a different sample rate, append it to your StreamUrl: wss://your-bot.example.com/media?sample-rate=16000. Supported values: 8000, 16000, 24000.
- Your bot can send audio back to the caller at any time during the call.
- When your bot wants to end the call, it closes the WebSocket connection.

For full protocol details, see the AgentStream documentation.

## Things to Keep in Mind

- StreamUrl must use ws:// or wss://. Plain HTTP URLs are not supported.
- The full StreamUrl (including any query string) must be under 600 characters.
- If you provide a StreamName, it must be 32 characters or fewer.
- Recording, status callbacks, and custom fields all work the same way as they do for regular outbound calls.
- This feature must be enabled on your account before use. Contact hello@exotel.com to get started.

## Need Help?

- Read the AgentStream guide to understand how Exotel streams audio to your bot.
- For API errors and response codes, see the Error Code Reference.
- Contact support at hello@exotel.com.

# 2.2. Connect Voice AI with Flow API

The Connect Voice AI to Flow API lets you place an outbound call that is controlled by an Exotel Flow. Flows let you design complex call experiences – bidirectional stream(Voice AI), unidirectional stream(Agent Assist), play messages, collect input, route to different outcomes – and can hand the caller off to a Agent at any point in the journey.

Use this API when your call needs logic beyond a direct bot connection: for example, playing a terms disclosure before connecting to a bot, or routing a caller to either a human agent or a bot based on their menu selection.

**Typical use cases:** Multi-step outbound Voice AI calls with Agent or human fallback, compliance-driven call flows, blended human + AI workflows.

## Before You Start

You will need:

- Your Exotel Account SID, API Key, and API Token (available from your Exotel Dashboard).
- An ExoPhone (virtual number) to use as the Caller ID.
- A Flow built in the Exotel dashboard. If your flow includes a bot, it should have a **Voicebot** or **Stream** applet configured with your bot's WebSocket URL.

## How It Works

1. You send an API request with the caller's phone number and the URL of an Exotel Flow.
2. Exotel dials the phone number.
3. Once the call is answered, the flow begins executing – playing audio, collecting DTMF, routing logic, etc.
4. When the flow reaches a Voicebot or Stream applet, Exotel opens a WebSocket connection to your bot and streams audio in real time.
5. The bot can speak to the caller and, when done, hand back to the flow or end the call.
6. Call status updates are sent to your configured callback URL.

## Setting Up Your Flow

To include a bot in your flow, add one of the following applets in the Exotel Flow builder:

| Applet | When to use |
|---|---|
| **Voicebot** | Connect the caller to an AI bot. Configure the WebSocket URL inside the applet. |
| **Stream** | Stream audio to a custom WebSocket endpoint for real-time processing. |
| **Connect** | Transfer the caller to a human agent (for bot-to-human handover). |
| **Passthru** | Pass call control to an external system mid-flow. |

The StreamUrl and StreamType for the bot are configured inside the applet – you do not pass them in the API request.

## API Reference

**Endpoint:**

```
POST /v1/Accounts/{AccountSid}/Calls/connect
```

**Base URLs:**

| Region | Base URL |
|---|---|
| Singapore | https://api.exotel.com |
| Mumbai | https://api.in.exotel.com |

Authentication uses HTTP Basic Auth with your API Key as the username and API Token as the password.

## Parameters

Required

| Parameter | Description |
|---|---|
| From | The phone number to call. Use international format (e.g., +919876543210). |
| CallerId | Your Exotel virtual number. |
| Url | Flow URL: **http://my.exotel.com/{your_sid}/exoml/start_voice/{app_id}** The App/Flow URL of the Exotel Flow that will control this call. Find this in your Flow settings in the dashboard App bazaar |

Optional

| Parameter | Required | Type | Description |
|---|---|---|---|
| CallType | No | String | trans for transactional calls. |
| TimeLimit | No | Integer | Max call duration in seconds. Max: 14400 (4 hours). |
| TimeOut | No | Integer | Ring timeout in seconds. |
| StatusCallback | No | String | Webhook URL for call status updates. |
| StatusCallbackEvents | No | Array | terminal, answered, or both. |
| CustomField | No | String | Metadata passed to the applet via Passthru. |

## Example Request

bash

```
curl -X POST 'https://<api_key>:<api_token>@api.exotel.com/v1/Accounts/<AccountSid>/Calls/connect' \  -d
'From=+91XXXXXXXXXX' \  -d 'CallerId=0XXXXXXXXXX' \  -d
'Url=https://my.exotel.com/<AccountSid>/exoml/start_voice/<flow_id>' \  -d 'Record=true' \  -d
'StatusCallback=https://your-server.com/callback'
```

Replace <flow_id> with the numeric ID of your Flow from the dashboard.

## Response

A successful request returns a Call object immediately. Use StatusCallback to track the call as it progresses through the flow.

json

```
{  "Call": {    "Sid": "a1b2c3d4e5f6...",    "Status": "in-progress",    "From": "+91XXXXXXXXXX",
"PhoneNumberSid": "0XXXXXXXXXX",    "Direction": "outbound-api",    "DateCreated": "2025-06-01 10:00:00",
"RecordingUrl": null  }}
```

| Field | What it means |
|---|---|
| Sid | A unique ID for this call. |
| Status | Current state of the call. |
| RecordingUrl | Link to the recording after the call ends (if recording was enabled). |

**Call status values:**

| Status | Meaning |
|---|---|
| queued | Call is being prepared. |
| in-progress | Call is active. |
| completed | Call ended normally. |
| failed | Call could not be placed. |
| busy | The number was busy. |
| no-answer | The number did not answer. |

## Common Flow Patterns

**VoiceBot Applet** → **Passthru Applet** → **Programmable Connect: Working with Connect Applet (Dynamic URL)** Place a Voicebot applet early in the flow, followed by a Connect or Transfer applet. The bot handles routine queries and escalates to a live agent when needed.

**Bot** → **Passthru Applet** Use a Passthru applet after the Voicebot applet to pass call control to your own telephony system for further handling.

**IVR** → **VoiceBot Applet handover** Design your flow with an IVR Menu applet first, then connect the appropriate option to a Voicebot applet. Callers navigate the menu, and the bot handles the selected intent.

**Bot with compliance disclosure** Start with a Greeting applet to play a required disclosure, then move to the Voicebot applet. The caller hears the disclosure before the bot begins the conversation.

## Things to Keep in Mind

- Your Flow must be created and active in the Exotel dashboard before making API calls.
- The Url parameter must point to a valid Exotel AppEngine flow URL. Custom external URLs are not supported for this parameter.
- The Voicebot and Stream applets must be configured inside the flow with the correct WebSocket URL. Do not pass StreamUrl or StreamType directly in this API call.

- Recording, status callbacks, and custom fields work the same way as they do for regular outbound calls.

## Need Help?

- Learn how to build flows with the Exotel Flow Builder guide.
- Configure your bot in a flow using the Voicebot Applet guide.
- For a direct bot connection without a flow, see the Connect to Bot API.
- Contact support at hello@exotel.com.

# 2.3. Applets

# 2.3.1. Connect Applet

## Connect Applet

The Connect applet routes an incoming call to a group or list of phone numbers. When a caller reaches this applet, Exotel dials the configured numbers and connects the caller to whoever answers first.

### Features

- **Flexible dialing** – Connect to one or multiple phone numbers
- **Two configuration modes**:
  - **Manual entry** – Enter phone numbers directly in the applet configuration
  - **Dynamic via URL** – Return numbers to dial from your application URL
- **Call recording** – Optionally record the connected conversation

### Configuration

1. In the Exotel Dashboard, open your call flow editor
2. Drag the **Connect** applet into your flow
3. Choose how to provide numbers:
   - **Enter directly**: Type phone numbers into the configuration
   - **Application URL**: Provide a URL that returns the numbers to dial

### Dynamic Number Selection

When using an application URL, your endpoint receives call details and should return the phone number(s) to connect. This enables dynamic routing based on:

- Caller ID (route VIP customers to dedicated agents)
- Time of day (route to different teams based on shifts)
- CRM data (route to the account owner)

For detailed documentation on the URL response format, refer to the Connect applet dial-whom documentation.

### Call Recording

When recording is enabled, combine the Connect applet with a Greeting applet to inform callers that their conversation is being recorded for quality assurance.

### Example Flow

```
Incoming Call → Greeting → Connect (to sales team)
```

# 2.3.1.1. Connect applet - Your tool to do complex call routing with Exotel

Every single customer of Exotel uses Exotel because they want complete control over how they route incoming calls and to whom. Exotel provides a variety of different options and ways to route your incoming calls. This article lists all possibilities, but before that, let's get some concepts out of the way:

**The team:** We have a hypothetical group called the "Support" group in which there are three members: Ajay, Bhaskar & Chaitanya in that order.

**Parallel Ringing(On Request Feature)**

With Exotel, it is possible to send an incoming call to Ajay, Bhaskar & Chaitanya at the same time. That is, Exotel can ring all their phones at the same time and connect the caller to the first person who picks the call up. Read more about parallel ringing

**Connect Configuration options:**

**Dial Whom:**

Choose the group to send this call to. Click on the "magnifying glass" to select a group. Alternatively, You can use the radio button to choose the "text box" section. In the text box, paste in a valid URL that can return text and valid HTTP headers. Refer here for the documentation of providing numbers to dial using your application URL.

The text box section is good & bad at the same time:

**The good:** You have the freedom to return a comma-separated list of numbers to dial. This is useful if you want to make a dynamic decision (based on the caller's number or "an input you collected using the Gather applet which the caller may have pressed"' in the previous step etc) on whom to call.

**The bad:** Exotel does not maintain the state of these numbers. That is - If another call comes and your script decides to send the call to the same set of numbers, Exotel will dial them and if the destination party has call waiting enabled, the caller will hear "The person you are trying to reach is on another call" etc. So - The user & user's state management is left to you to handle.

**NOTE: This text box option for pasting URL will only appear in Connect applet if you have submitted your KYC documents successfully.**

**Call recording**

If you select this option, an entry in the inbox will also have a recording that you can play & listen to at a later point. If this is not checked, then there will still be a line item in your inbox without the recording. Read more about call recordings here.

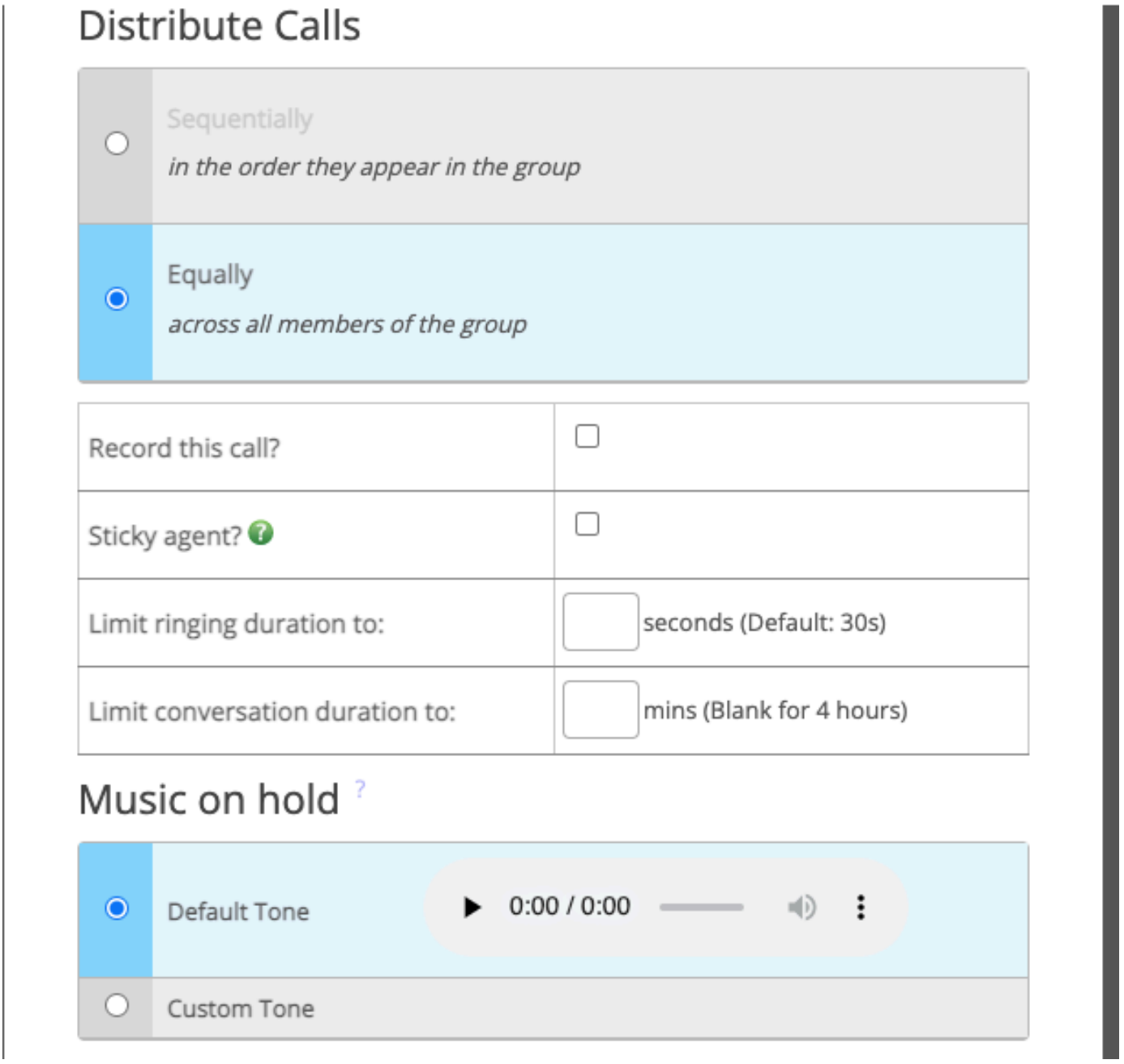
**Call routing - default option**

The default call routing option in a group is the order in which the names appear once you click on the group name on the Co-workers & Groups page. That is, Every incoming call will be sent to Ajay first and if he does not pick up, to Bhaskar and then to Chaitanya. The default option respects whether the user is available & free. That is, If Ajay has turned himself "off" from the exotel system, then the call goes straight to Bhaskar. If Bhaskar is already talking to someone, then to Chaitanya.

**Distribute calls equally to Group members:**

This option is similar to "Round robin". In the round robin, If the first call today goes to Ajay, then the 2nd call goes to Bhaskar, the third to Chaitanya, etc. The point of the round-robin is to equally distribute the load of calls to all members in a group. In the example above, if Bhaskar cannot pick up when the second call comes in, the call is routed to Chaitanya. The third call then goes to Ajay and the fourth to Bhaskar. It is assumed that an agent who is "ON" would be free to pick up all calls. If you don't check this option, then it will be the

above "default" option. Click here to know more about this feature.




**Sticky Agent**

When an incoming call from a new user (Say Dinesh) is picked up by Bhaskar, then every time Dinesh calls, The call will always first go to Bhaskar. If Bhaskar is busy talking to someone else or is currently switched off from Exotel, then the call follows the usual routing configurations. This option is useful in assigning "account managers". That is, a single person responsible for a customer.

Sticky Agent will also work in an outbound scenario. Consider the above example where Bhaskar has used the Voice API (v1/v2/v3) to reach out to Dinesh. In this scenario, Exotel would have first dialed out to Bhaskar, who picks up, and then Exotel dials out to Dinesh.

Irrespective of whether Dinesh picks up or not, Exotel will now ensure a mapping exists between Dinesh and Bhaskar for a fixed duration of 30 days. Dinesh will be able to connect back on the number and reach out to Bhaskar itself (the sticky agent) if a call flow attached to the same VN (that was used in the Voice APIs ) has a connect applet with Bhaskar as one of the agents in the Agent Group.

**Number of seconds to ring an agent**

This is the amount of time to ring Ajay's phone number before moving on to Bhaskar. This time period starts from the time the server starts to dial (and not the time Ajay receives the call). There will be a 2 to 3-sec delay before the call lands on Ajay's phone (We all experience this delay. When you dial your friend's number, you will hear "beep, beep, beep" before it goes tring tring). The default is 30 secs. We recommend that you do not reduce this to less than 10 secs as Exotel might move on to Bhaskar even before Ajay realizes that he is getting a call on his number. The "beep beep" network slowness depends on the system load of not only Exotel (which we are constantly monitoring) but also the load and congestion in the source & destination operator networks.

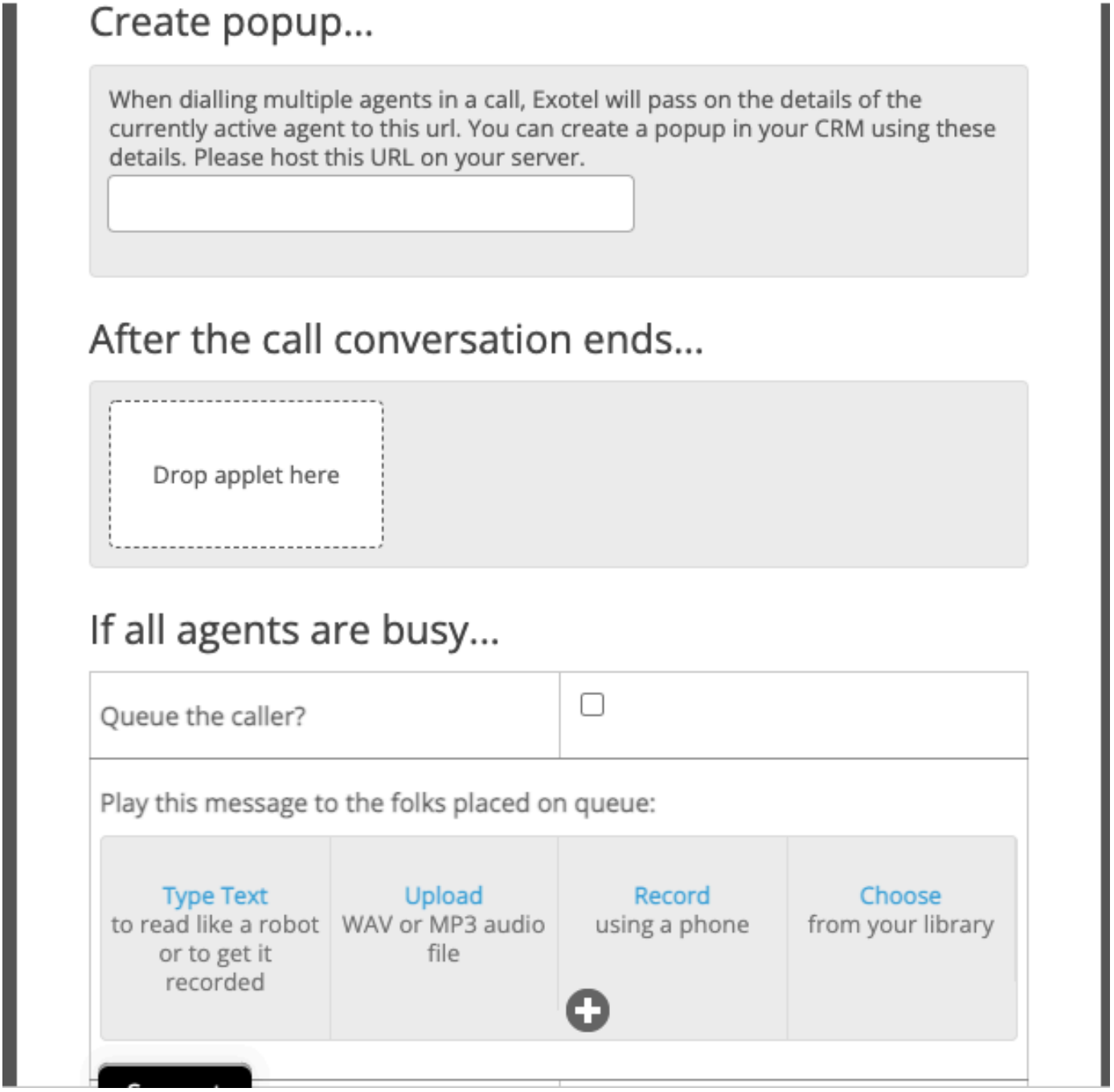
**Limit call duration**

This is the length of the time period up to which the call can be active. The default is 4 hours and I can tell you that no one has ever crossed that limit yet. We recommend that you leave this blank. This option is particularly useful when you want to control the length of a call. For ex: If you have paid support and let's say the user has paid for a 30-minute slot, then you could use this to cut the call at the end of 30 minutes. Currently, Exotel does not inform the caller or the callee that their time period is over. It simply hangs up!

**Music On Hold (Ringing Tone)**

This is the music that will be playing when we dial the agent's number (Note: This is not the hold music that plays when you put the call on hold on your handset)

There are more options in connect:




**Create Pop-Up ( It triggers a call pop-up notification on your endpoint)**

This is particularly useful to dynamically send information to one of your servers or systems. During the call option sends the parameters from number(customer number), to number(agent number), and caller ID(Exotel number); more about these parameters can be found here During the call.

**After the call conversation ends.**

There are a few things you can do when Ajay and Dinesh finish talking:

- Send an SMS to Dinesh saying "Thank you for calling Mannar & Company. We hope to hear from you again!". Drag/Drop the SMS applet to achieve this.
- Send an Email to the "Support" Group with the details of the call. Read more about the email applet.
- Pass the information through to your system/server using Passthru.

If you leave this option empty, then the call will simply hang up (which is what you really want in most cases). Just to be clear, drag/drop the "Hangup" applet.

**Queue the caller**

If all the agents are busy, You can put the caller in a queue and make him wait. Exotel will wake up every 10 seconds to check all callers in your queue and if there is a free agent who can talk to them. Read more about Call queueing here.

**If Nobody answers**

Exotel will look into your configuration in this option if no one picked up the call in your group. To be specific, this option will be looked up if any of the following options are true:

a) There were agents in the group, some of them were free, Exotel sent the call to them but they did not pick up.

b) There were no free agents in the group at the time of the call

c) There were no agents at all in the group!

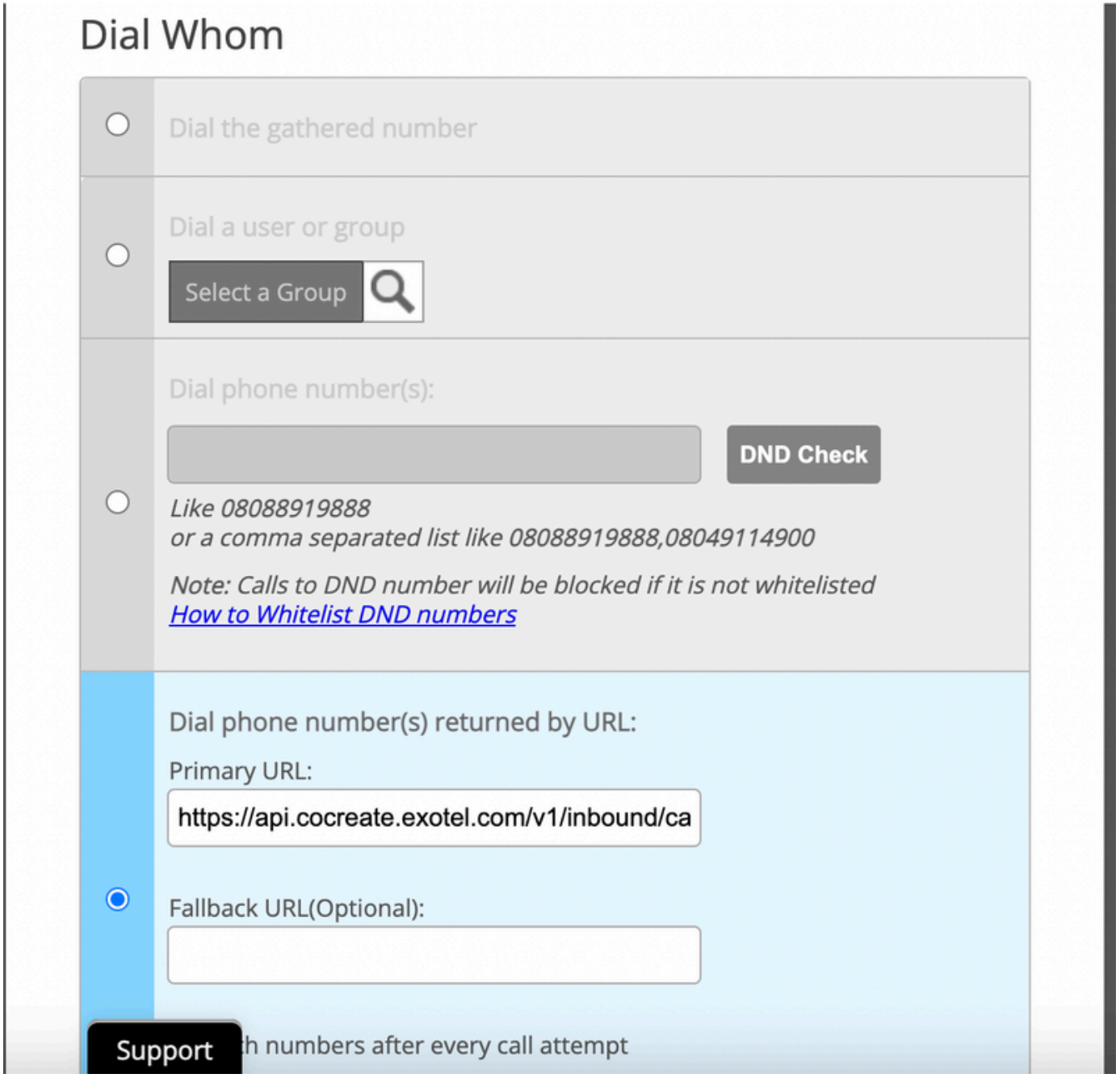
d) You had enabled the queue & after having placed the caller in the queue, the queue time out exceeded.

Choice 1: Say sorry & Hangup up plays this message to the caller & hangs up.

Choice 2: You could drag/drop another applet. For ex, a Voicemail or even an SMS applet that sends an SMS informing the customer that all three people were busy.

# 2.3.1.2. Dial using number from a URL

In the Connect applet, in the "Dial Whom" section, there is an option to "Dial phone number (or phone number returned by this URL)". You can specify a URL in the text box. When the text box contains a URL, Exotel will make an HTTP `GET` request to that URL, and the URL should return a number in the HTTP response. This is the number that Exotel will try dialing, and connecting the caller to.




In the 'Dial Whom' section of connect applet, you can configure an incoming call to a group, add static numbers or provide a URL from where you can get a number to be dialed by the way of an HTTP GET request.

The following are the query parameters of the GET request if you choose to return numbers from a URL. Note that only some of this list may be passed to your application - depending on what stage of the flow you have placed the applet.

**Query Parameters:**

| Parameter Name | Description |
|---|---|
| CallSid | Unique identifier of the call |
| Callfrom | In case of an outgoing call, it'll be set to the number from which the call is made. In case of an incoming call, it'll be set to the number from which the call is received |
| CallTo | In case of an outgoing call, it'll be set to the number being dialed out. In case of an incoming call, it'll be set to the number where the call landed. |
| CallStatus | The status of the call depends on what stage it is at.<br>Possible values: 'queued', 'ringing', 'in-progress', 'completed', 'busy', 'failed', 'no-answer', 'canceled' |
| Direction | The direction of the call. Possible values: 'incoming' or 'outbound-dial' |
| Created | Timestamp when the call is created |
| DialCallDuration | Value in seconds from the time the call is triggered to the second leg of the call till it is over (including conversation time). This value can be set to zero depending on the previous applet and if there's no second leg in the call flow. |
| StartTime | Timestamp when the call is started |
| EndTime | 1970-01-01 05:30:00 // Unix time (also known as POSIX time or epoch time)<br>Note that this would be a constant value and you may instead trigger our Call details API a few minutes after the call has been completed to get accurate information |
| CallType | Scenario<br>Value<br>IVR only, no connect applet<br>call-attempt<br>Call conversation happened<br>completed<br>The client hung up during connect applet<br>client-hangup<br>Connect applet, no agent picked up<br>incomplete<br>Went to voicemail applet<br>voicemail |
| DialWhomNumber | Displays the number of the agent who was dialed to last |
| From | In case of an incoming call, it is the number of the caller. In the case of an outgoing call, it is the number of the first leg of the call. |
| To | In case of an incoming call, it is the ExoPhone on which the call landed. In case of an outgoing call, it is the number on which the call was made to. |
| CurrentTime | Current server time (format : yyyy-mm-dd hh:mm:ss) |

*These parameters will be passed if certain conditions are met as described below:

| Parameter Name | Description |
|---|---|
| DialCallStatus | This will denote what happened with the second leg of the call for subsequent requests to your connect application URL (in case of multiple attempts) or if the previous applet itself was “connect”.<br><br>This is to provide you information about the call status of the previous attempt particularly in case your application URL returns comma-separated numbers (N1, N2 ... )<br><br>Possible values: 'completed', 'busy', 'no-answer', 'failed', 'canceled' |
| digits | ‘digits’ will be passed if there was a 'Gather' or ‘IVR’ applet before this applet and will be equal to the input digits that were entered. NOTE: This parameter comes with a double quote (") before and after the number. You'll have to trim() this parameter for double quotes (") to get the actual digits. |
| CustomField | If the call was initiated via API, the value that was passed in CustomField in the API call will be set here. |
| RecordingUrl | This will be populated if the previous applet was "voicemail". It will contain the URL of the voicemail recording. There could be a delay before the recording can be accessed depending on the length of the recording file. |

NOTE: Some of these parameters like digits will only be passed in the first request of ‘Dial Whom’ and subsequent requests (if any) will contain the result of the previous connect applet attempt.

**Deprecated parameters:**

The following parameters have been deprecated and are no longer supported. Some of these values might be passed inadvertently and applications should not rely on them anymore.

| RecordingAvailableBy |
|---|
| ForwardedFrom |
| ProcessStatus |
| tenant_id |
| flow_id |
| numbers |

**Expected Response:**

Your application URL response must:

- have only the phone number in the response body and nothing else.
- have comma-separated numbers, in case you wish to provide multiple numbers.
- content-type to be set as ‘text/plain’ with HTTP code set as 200

For example:

`+919900XXXXXX`

`+919900XXXXXX,+919800XXXXXX`

Note: Please provide numbers in E.164 format.

If the previous attempted number didn't connect, **'Fetch Numbers after every call attempt'** comes into play. This option will be checked by default as shown above and your application URL will be requested subsequently after an attempt for giving the flexibility of dynamically controlling the number(s) to be dialed out. You can return the same set of numbers as well, and Exotel will attempt each unique number. Once, all the returned number(s) have been attempted at least once (in case no one picks up), it'll transition to the next applet. You can also choose to uncheck this option. In this case, your application URL can be configured to return number(s) to be dialed in a single request and Exotel will attempt all the numbers provided one by one until someone picks up or all attempts are exhausted. We recommend you to use this method unless you have a use case to dynamically return numbers based on each call attempt.

**Fallback URL**

This parameter is optional and is valid if you configure the connect applet to return numbers using your application endpoint (set under Primary URL).

Scenarios when we will request your Fallback URL to return numbers:

- If your primary URL responds with a non-200 HTTP code (like 4XX, 5XX)
- If your primary URL is not reachable and times out (doesn't respond within 10 seconds)

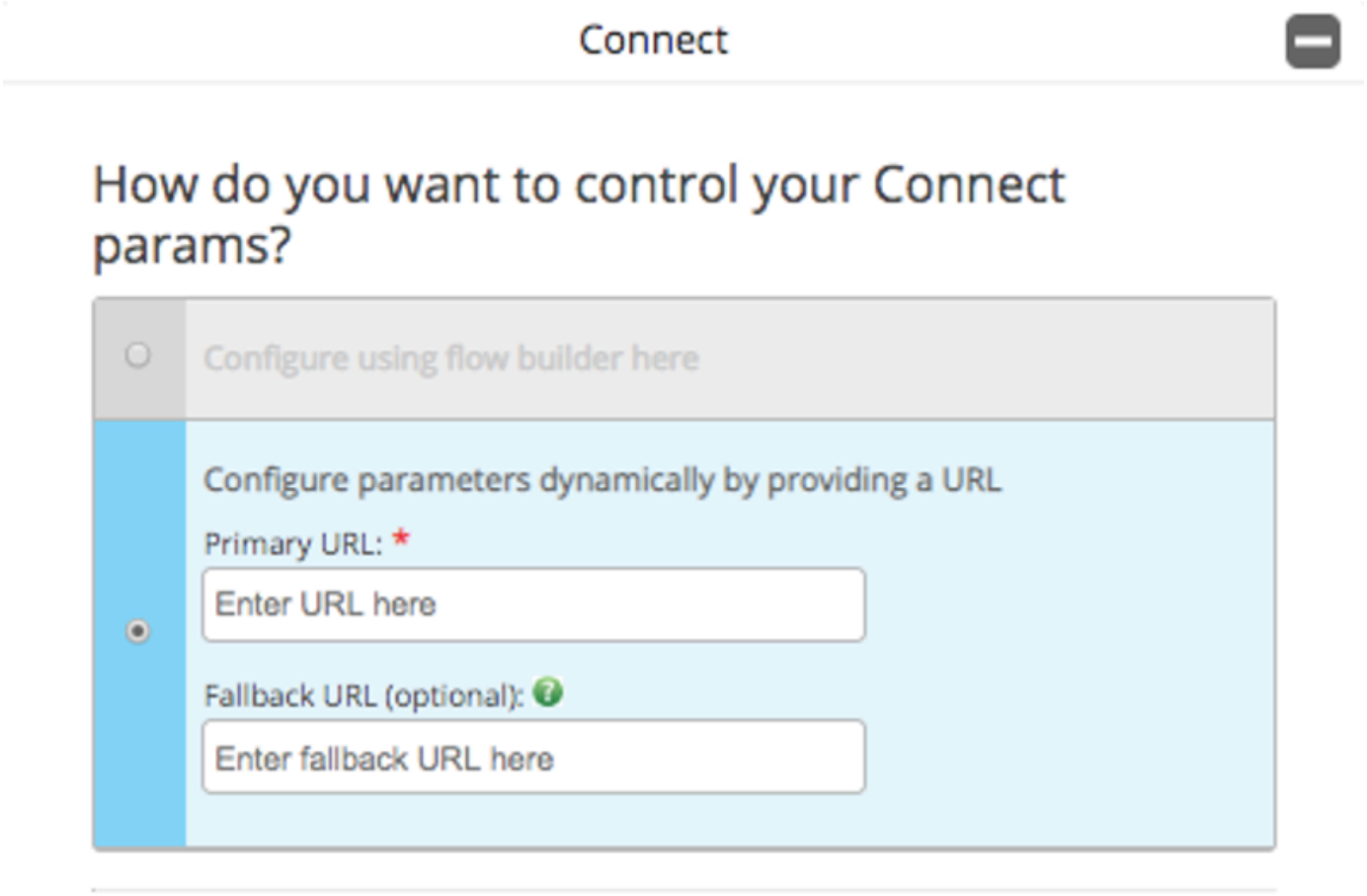
Scenarios where Fallback URL will NOT be triggered:

- If your application URL responds with HTTP code 200 but doesn't return any numbers
- If your application URL responds with HTTP code 200 but returns invalid numbers or values

If you have any questions or concerns, please connect with us using the chat widget on your Exotel Dashboard or Whatsapp us on 08088919888

NOTE: This option will be available only if your KYC docs have been submitted & approved.

# 2.3.1.3. Programmable Connect: Working with Connect Applet (Dynamic URL)




You can choose to configure the Connect applet parameters using the flow builder itself or control it dynamically through a URL. However, you will need to configure the transitions (next applet) while building the flow irrespective.

If you choose to configure the parameters dynamically using your application URL, you can set a 'Primary URL' for handling the requests. You can also set a 'Fallback URL' (optional) which will be contacted in case something goes wrong with your 'Primary URL'.

***Request***

If an application URL is set for Connect applet, Exotel will make a GET request to the URL with the call details as URL-encoded HTTP query parameters.
The following are the parameters of the GET request. Note that only some of this list may be passed to your application - depending on what stage of the flow you have placed the Connect applet.

Header:

| | |
|---|---|
| Exotel-Version | This value will indicate the version of connect applet parameters against which your endpoint's response will be validated.<br><br>Current Version: 1.0 |

Query Parameters:

| Parameter Name | Description |
|---|---|
| CallSid | Unique identifier of the call |
| CallFrom | In case of an outgoing call, it'll be set to the number from which the call is made. In case of an incoming call, it'll be set to the number from which the call is received |
| CallTo | In case of an outgoing call, it'll be set to the number being dialed out. In case of an incoming call, it'll be set to the number where the call landed. |
| Direction | The direction of the call. Possible values: 'incoming' or 'outbound-dial' |
| Created | Timestamp when the call is created (format : yyyy-mm-dd hh:ss) |
| DialCallDuration | Value in seconds from the time call is triggered to the second leg of the call till it is over (including conversation time). This value can be set to zero depending on the previous applet and if there's no second leg in the call flow. |
| StartTime | Timestamp when the call is started (format : yyyy-mm-dd hh🇲🇲ss) |
| EndTime | 1970-01-01 05:30:00 // Unix time (also known as POSIX time or epoch time)<br><br>Note that this would be a constant value and you may instead trigger our Call details API a few minutes after the call has been completed, to get the accurate information |
| CallType | |
| DialWhomNumber | Shows the number of the agent who was dialed to last |
| flow_id | Flow id associated with the call |
| From | In case of an incoming call, it is the number of the caller. In case of an outgoing call, it is the number of the first leg of the call. |
| To | In case of an incoming call, it is the ExoPhone into which the call came. In case of an outgoing call, it is the number to which the call was made. |
| CurrentTime | Current server time (format : yyyy-mm-dd hh:ss) |

*These parameters will be passed if certain conditions are met as described below:

| Parameter Name | Description |
|---|---|
| DialCallStatus | This will denote what happened with the second leg of the call if the previous applet was “connect”.<br><br>Possible values: 'completed', 'busy', 'no-answer', 'failed', 'canceled' |
| digits | ‘digits’ will be passed if there was a 'Gather' or ‘IVR’ applet before this applet and will be equal to the input digits that were entered. NOTE: This parameter comes with a double quote (") before and after the number. You'll have to trim() this parameter for double quotes (") to get the actual digits. |
| CustomField | If the call was initiated via API, the value that was passed in CustomField in the API call will be set here. |
| RecordingUrl | This will be populated if the previous applet was "voicemail". It will contain the URL of the voicemail recording. There could be a delay before the recording can be accessed depending on the length of the recording file. |

***Response***

This is the response Exotel will expect to the `GET` request from your Connect Application URL. The response will decide what parameters will be set while executing Connect during the call flow.

Response Header:

| | |
|---|---|
| Content-Type | application/json |

Sample:

```
{
"fetch_after_attempt":false,
"destination": {
"numbers":["+919812345678"]       },      "outgoing_phone_number":"+918047115777",
"record": true,
"recording_channels":"dual",
"max_ringing_duration":45,
"max_conversation_duration":3600,
"music_on_hold":
{
"type":"operator_tone"
},
"start_call_playback":
{
"playback_to":"both"
"type":"text",
"value":"This text would be spoken out to the callee"
}
}
```

Generic

*Explanation:*

| Parameter | Mandatory/ Optional | Description |
|---|---|---|
| fetch_after_attempt | Optional; default = false | This parameter will indicate if, after each unsuccessful dial attempt within connect, the parameters should be fetched again including destination numbers.<br><br>`false` will indicate, dial attempt to happen based on the initial response. No subsequent hits to the URL.<br><br>`true` will indicate if connect parameters including a number to dial should be fetched again (hit the configured URL again) if the dial attempt is unsuccessful.<br><br>NOTE: If 2 subsequent fetch results contain exactly the same set of destination numbers, Exotel will not make any subsequent attempts even if fetch_after_attempt is set to true.<br><br>Request:<br>GET /<customer-url><br><br>Apart from standard request params, it'll include <connect> params from the previous dial attempts.<br><br>Response:<br><Same as above> |
| destination | Mandatory | Indicates the destination(s) to dial out.<br><br>numbers<br>An array of numbers to be dialed out in E.164 format.<br><br>The dial will happen in the order they appear in the array.<br><br>Sample:<br><br>"destination": {<br>"numbers": ["+622131921111", "+622131921112"]<br>}<br><br>Sample in case of Inbound SIP trunking<br>{<br>"fetch_after_attempt": false,<br>"destination": { "trunk": "trunk-2134" },<br>"custom_params": "param1=value1&param2=value2",<br>"record": true,<br>"recording_channels": "dual"<br>} |

| Parameter | Mandatory/ Optional | Description |
|---|---|---|
| | | |
| outgoing_phone_number | Optional; default = Incoming ExoPhone | ExoPhone to be used for dialing out in E.164 format.<br><br>Validations:<br>• The ExoPhone added should be present in your account.<br>• Restrictions will depend on telecom regulations i.e. another call can only be dial-ed out from the same telecom circle/region. For example, outgoing ExoPhone cannot be set to Delhi where Incoming ExoPhone is in Bangalore.<br>• Both the first leg and the second leg ExoPhones can be connected on the same server.<br><br>NOTE: If ExoPhone is not present in your account or the ExoPhone is unable to dial out i.e. if the above validation fails, then the call would be dial-ed out using the same ExoPhone as the first leg (where the incoming call landed).<br><br>**NOTE: Consult with exotel support to use this feature.** |
| record | Optional; default = false | true/false; Record the call or not |
| recording_channels | Optional; default = single | To record the caller and callee in separate channels in the recording file.<br><br>Possible values:<br>• single<br>• dual |
| max_ringing_duration | Optional;<br><br>default = 30 | Value in seconds to limit the ringing duration. This can be increased up to 60 seconds. |
| max_conversation_duration | Optional;<br><br>default = 900 (15 minutes) | Value in seconds to limit the conversation duration to. This can be increased up to 75 minutes (4500 s). |
| music_on_hold | Optional;<br><br>default = default_tone | Possible Values:<br>• default_tone: Exotel default music on hold as present here.<br>• operator_tone: Audio returned by the operator on the dialing channel as is.<br>• custom_tone: Audio URL as provided in the response. |

| Parameter | Mandatory/ Optional | Description |
|---|---|---|
| | | Sample Values:<br><br>{<br>“type”: “default_tone”<br>}<br><br>{<br>“type”: “operator_tone”<br>}<br><br>{<br>"type": "custom_tone",<br>"value": "<audio_url>"<br>} |
| parallel_ringing | Optional; | Use this option to dial the numbers in parallel (simultaneously). This will dial all the numbers returned under the destination parameter.<br><br>"parallel_ringing": {<br>"activate": true,<br>"max_parallel_attempts": 5<br>}<br><br>max_parallel_attempts value can be between 1-10. Default: 5<br><br>*Please note this feature is chargeable and consult with your Account Manager or email to hello@exotel.in before using this parameter. |
| dial_passthru_event_url | Optional; | The URL passed here will be requested for dial start and dial end events. For more details on request, refer to this. |
| start_call_playback | Optional; | Play a recording to the number that is being called<br><br>{<br>“playback_to”: “both”,<br>“type”: “audio_url”,<br>“value”: “http://...mp3”<br>}<br>OR<br>{ |

| Parameter | Mandatory/ Optional | Description |
|---|---|---|
| | | “playback_to”: “both”,<br>“type”: “text”,<br>“value”: “hello, this is a sample text”<br>}<br>OR<br>{<br>“playback_to”: “callee”,<br>“type”: “audio_url”,<br>“value”: “http://...mp3”<br>}<br>OR<br>{<br>“playback_to”: “callee”,<br>“type”: “text”,<br>“value”: “hello, this is a sample text”<br>}<br>Configuration for audio file supported in this playback are:<br>Sample Rate = 8 kHz<br>Bit depth = 16 bit<br>Bit rate = 128 kbps<br>Channel = mono<br>File Format = wav<br><br>Note: If the file name returned in case of `audio_url` is the same, it will be cached by our servers. Kindly, provide dynamic file names if the audio to be played is different each time. |

*Above set of parameters can also be controlled through the dashboard if one opts to configure the Connect applet using the flow builder instead of the Application URL.

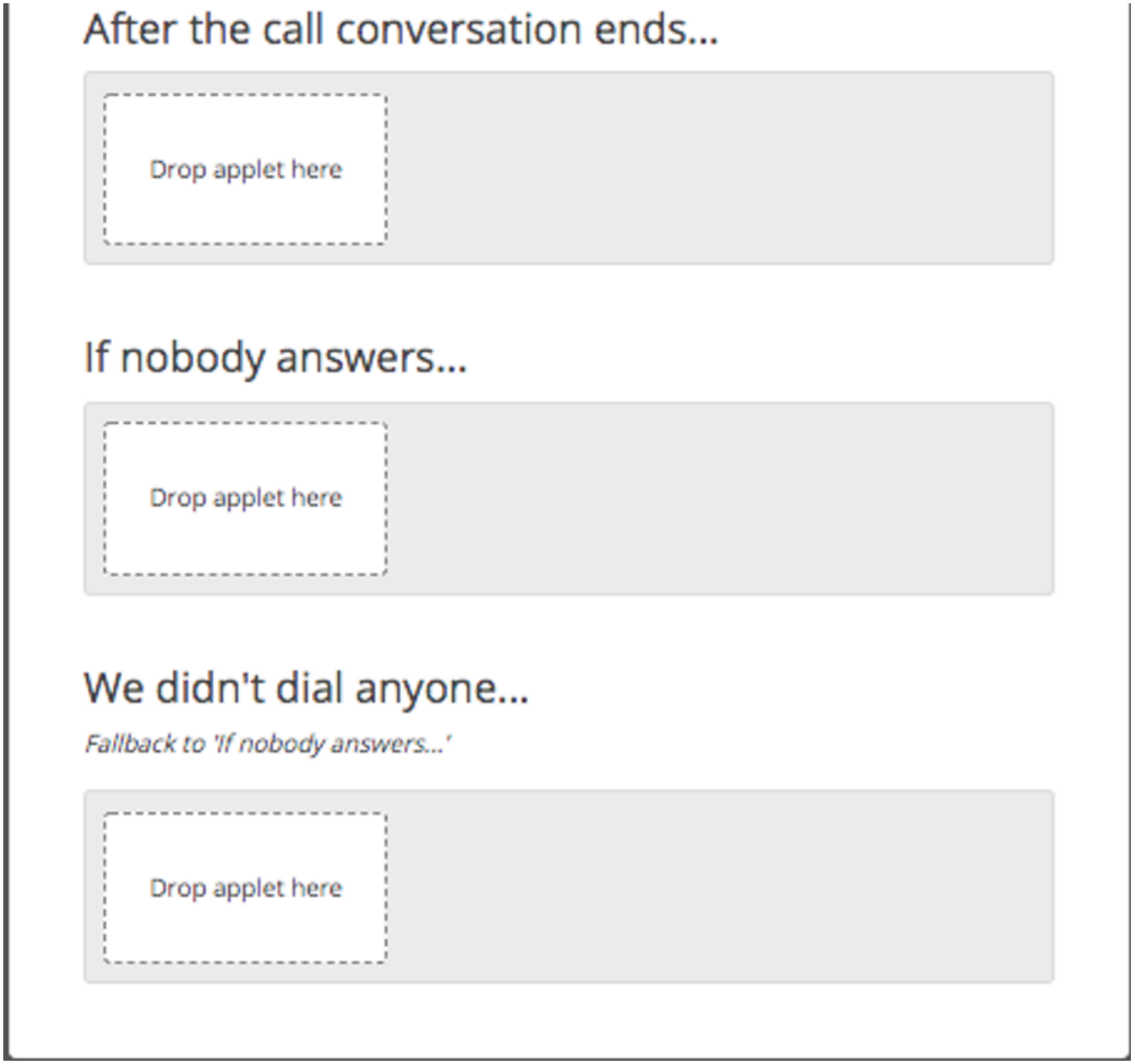
Cases, where Fallback URL will be triggered, are:

- Primary URL does not return HTTP 200 OK
- Primary URL doesn’t return within the timeout period (5 seconds)
- Primary URL returns invalid response:
  - Content-Type should be application/JSON
  - Mandatory parameters are present
  - audio_url / text should be HTTP/HTTPS and returns 200 HTTP code

**Transition to next applet**

---

Below transitions are to be set in the call flow builder to decide what to execute next.




Scenarios:

- After the call conversation ends: The applet set here will be triggered if a conversation occurs.
- If nobody answers: The applet set here will be triggered if we dial the number and conversation doesn't occur. If no applet is present in "we didn't dial anyone", it will fallback to this.
- We didn't dial anyone: The applet set here will be triggered if we don't dial. Cases when this can occur:
  - If both Primary and Fallback URL endpoint times out or returns non-2xx response code.
  - If both Primary and Fallback URL endpoint doesn't return a valid payload.
  - If the number to dial returned is in invalid format.
  - If an empty number (destination) is returned in the response

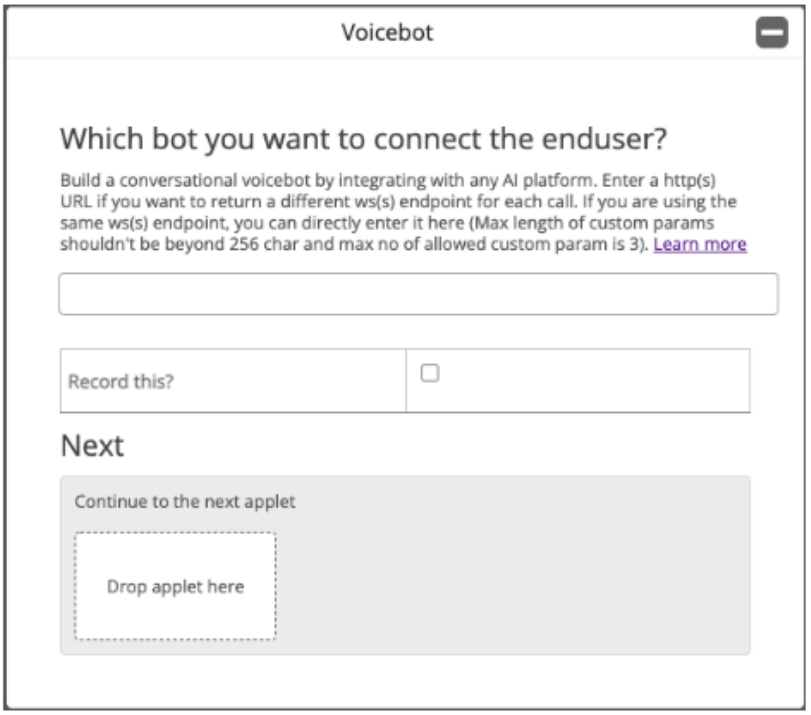
# 2.3.2. Voicebot Applet

**Configuring a Bidirectional Stream- Voicebot Applet**


Bidirectional Streaming

Bidirectional Streams allow two-way flow of voice data over a websocket. Exotel would send the voice data of the caller to a websocket endpoint. The endpoint can return back voice data back on the websocket and Exotel would play it out to the caller. The primary use case for this is to enable building intelligent conversational bots that will help you optimize your workforce WebSocket

You can enable bidirectional streaming on a call flow using the Voicebot Applet.




The applet takes 6 parameters:

**1. URL-** This is the URL to which Exotel will stream the voice media. You can either specify a wss endpoint or a https endpoint. If you specify a http/https endpoint, Exotel expects the https endpoint to return a wss url in its response. This is to allow

a. Dynamic endpoints for the same call flow
b. Have dynamic custom parameters that can be passed to the websocket endpoint to handle any application-specific customisation.

There are two ways to put this:
a. Static method: ws(s) endpoint can be entered here, but it will remain the same for every call that you are going to make using this flow. eg: `ws://127.0.0.1:5001/media`
b. Dynamic method: We can enter http(s) URL which can return different ws(s) endpoints based on the use case.
eg: `https://yourdomain.com` this URL must return a ws(s) endpoint

**2. Authentication Options for WSS Endpoints-** When configuring your WSS endpoint for the Voicebot Applet, Exotel supports the following authentication methods:

**IP Whitelisting (Basic Setup)**
Secure your WSS connections by whitelisting Exotel's outbound IP ranges. This avoids the need for credential exchange.

Reach out to hello@exotel.com to get the range of IPs.

**Basic Authentication**
Developers specify credentials in the WSS URL, but Exotel transmits them securely in headers during the connection.

- **WSS URL Configuration (Developer Input):**

```
wss://<API_KEY>:<API_TOKEN>@stream.yourdomain.com/<stream-path>
```

- **What Exotel Sends (Header):**

```
Authorization: Basic base64 (API_KEY:API_TOKEN)
```

This approach ensures compatibility and prevents credentials from being exposed in transit URLs.

**3. WSS Sample Rate Parameters-**Your WSS endpoint should support configurable sample rates. Exotel's Voicebot Applet can define the required rate as a query parameter.

- **8 kHz (Standard PSTN Quality – Default):**
  - wss://your-domain.com/?sample-rate=8000
- **16 kHz (Enhanced Quality):**
  - wss://your-domain.com/?sample-rate=16000
- **24 kHz (HD Quality):**
  - wss://your-domain.com/?sample-rate=24000

**Default Behaviour:** If no sample rate is explicitly defined, Exotel will use **8 kHz** as the default.

**Best Practice:** Use 16 kHz for most voicebot integrations (balanced quality vs. bandwidth). Use 8 kHz only when interworking with legacy PSTN and 24 kHz for HD audio experiences.

**4. Custom parameters(Optional)** - Custom params along with the endpoint. There are some validations that need to follow while providing custom params.

a. Maximum number of custom params that are allowed is 3.

b. Format of these params will like :

`ws://127.0.0.1:5001/media?param1=value1&param2=value2&param3=value3` (In Dynamic case, http(s) should return ws(s) URL in above format)

c. Total length of the params( bold part in above url ) shouldn't be more than 256 characters.

**5. Record** - The checkbox gives an option to record the conversation and generate a recording URL available in passthru applet after voicebot applet.

**6. Next Applet** - In the case of a Bi-Directional Stream. The stream can end if the call is disconnected or the websocket is closed or the stream is explicitly stopped by the client. In the case of Bi-Directional Stream you do not need to add a explicit "Stop" Stream applet since the stream is automatically closed before executing the next Applet. The call flow proceeds to the next applet configured.

**Video WalkThrough**

You can find a quick walkthrough of a sample flow here.

**Protocol**

Communication between Exotel and customer endpoint happens over websocket connection.

**Websocket messages - From Exotel**

Each message in the websocket will be sent/received as a JSON string. Following are the types of messages that are sent:

- - Connected
- - Start
- - Media

- - DTMF
- - Stop
- - Mark (Only in Bidirectional)
- - Clear

**Connected message:**

**After websocket connection is established, this message will be sent.**

JSON

```
{
"event" : "connected",
}
```

**Start message:**

Start message will contain information about the stream parameters. It will be sent only once, right after the connected message. The custom parameters are picked from the URL configured in the Stream Applet. If you had mentioned the URL as

`wss://yourstream.service.com?queuename=premium&product=radio`

queuename and product would be passed in as keys with premium and radio as values.

JSON

```
{
"event" : "start",
"sequence_number" : 1,
"stream_sid" : "<stream sid>",
"start" : {
"stream_sid" : "<>",
"call_sid" : "",
"account_sid" : "",
"from" : "",
"to" : "",
"custom_parameters" : {
"Key1" : "value1",
"key2" : "value2"
},
"media_format" : {
"encoding" : "<>",
"sample_rate" : "<>",
“bit_rate” : “<>”
}
}}
```

**Media message:**

This message encapsulates the audio packets.

JSON

```
{
 "event" : "media",
"sequence_number" : 3,
"stream_sid" : "<stream sid>",
"media" : {
"chunk" : 2, "timestamp" : "10", "payload" : "<>"
}
```

media.chunk : chunk of the message
media.timestamp : Timestamp in milliseconds from the start of the stream.

Media in the payloads are sent in raw/slin (16-bit, 8kHz, mono PCM (little-endian)) encoded in base64. The same is expected from the client in the case of bi-directional streams (In the case of a Voice Bot) to be played back to the human.

**DTMF message:**

DTMF message is sent when the digits are pressed by the user once the connection with websocket is established. This is supported only for bidirectional streaming in voicebot applet.

JSON

```
{
"event" : "dtmf",
"sequence_number" : 1,
"stream_sid" : "<stream sid>",
"dtmf" : {
"duration" : "<duration in ms>”,  "digit": <>, }
}
```

**Stop message:**

Stop message is sent when the stream is stopped or the call has ended.

JSON

```
{
 "event" : "stop",
"sequence_number" : 10,
"stream_sid" : "<stream sid>",
"stop" : {
"call_sid" : “<>”,
"account_sid" : "<>",
 “reason” : “stopped or callended"
}
}
```

**Mark Message:**

Mark message is used only in bidirectional streaming to track media when it is completed.

JSON

```
{"event" : "mark",
"sequence_number" : 15,
"stream_sid" : "<stream sid>",
"mark" : {
"name" : "<label>”, }
}
```

**Websocket messages - To Exotel**

These messages will be used only in bidirectional streaming.

**Mark Message:**

Mark message is used only in bidirectional streaming to track media when it is completed. You can send a mark event message after sending a media message to request a notification when the audio that you have sent has been processed. You'll receive a mark event message with a matching name from Exotel when the audio is processed.

JSON

```
{"event" : "mark",
"sequence_number" : 15,
"stream_sid" : "<stream sid>",
"mark" : {
"name" : "<label>”, }
}
```

**Media message:**

This message encapsulates the audio packets.

JSON

```
{
 "event" : "media",
"sequence_number" : 3,
"stream_sid" : "<stream sid>",
"media" : {
"chunk" : 2, "timestamp" : "10", "payload" : "<>"
}
```

**Clear message:**

Clear message is used to clear the audio data that was sent before but not yet played. An example situation wherein it will be useful,

Developing human-like bots which can guess what he/she is going to say and send audio accordingly even before he/she completes it. When the guess goes wrong, we can clear that audio using a clear message.

```
{ "event": "clear", "stream_sid": "<stream sid>",}
```

Note: For Clear Event to work effectively, it is advisable to send media in smaller chunks. For instance, if two media messages of 5 seconds are sent to us, followed by Clear Event at the 3rd second, and the first message has already been picked up and played, the Clear event will only apply to the second media message. Therefore, sending media in smaller chunks can help prevent confusion and ensure that Clear Event works as intended.

**Media Format**

Media in the payloads are sent in raw/slin (16-bit, 8kHz, mono PCM (little-endian)) encoded in base64. The same is expected from the client in the case of bi directional streams to be played back to the caller.

**Event Struct - Field Reference Table**

| Field Name | Type | JSON Key | Optional? | Description / Notes |
|---|---|---|---|---|
| Event | string | event | No | Type of the event: "start", "media", "stop", "dtmf", etc. |
| StreamSID | *string | stream_sid | Yes | Unique identifier for the stream session |
| SequenceNumber | *string | sequence_number | Yes | Ordering number for incoming media chunks |
| Start | *Start | start | Yes | Present when the event is a start event |
| Media | *Media | media | Yes | Present when the event is a media event (contains audio data) |
| Stop | *Stop | stop | Yes | Present when the event is a stop event |
| Mark | *Mark | mark | Yes | Present when the event is a mark event |
| Dtmf | *Dtmf | dtmf | Yes | Present when the event is a dtmf (key press) event |

**Sample Code**

https://github.com/exotel/Agent-Stream and

https://github.com/exotel/Agent-Stream-echobot

**Simulator to make streaming calls with dummy bot**

```
https://github.com/exotel/voice-streaming/commit/d4696f3cb13fb5d75ac46bed2aaaa2afababa10f#diff-217ed82f87b78b399e304b07a159b27dff327c0f83adf4a2fc30b03bcbf84b01
```

**Chunk size window for Bidirectional streaming use case:**

Minimum chunk size: 3.2k [100ms data]

Maximum chunk size: 100k

Chunk size should always be in multiple of 320 bytes.

1. If the size is less than the minimum size, we may face audio issues due to network jitters.

2. If the size is greater than 100k, it might result in timeouts

3. if the size X [for ex 4096] which is not in multiple of 320 Bytes: In this case, the last packet will be of lesser size than 320 Bytes, & platform will wait for 20ms before sending next chunk. i.e. sending less amount of data than wait time, then this might result into audio gaps in between.

**Limitations**

1) When you start a Unidirectional Stream, the stream is forked immediately. In case you have a Connect applet post this, the audio stream, even when Exotel dials out multiple agent,s would be relayed (ringing). The client will have to handle this to filter only relevant parts of the stream. This limitation will be fixed in future releases

2) The number of Custom Parameters that can be passed in the START message is 3

3) The stream is sent as a mono-channel raw audio format, and the client will have to handle speaker diarization

If you have any questions or concerns, please connect with us using the chat widget on your Exotel Dashboard or WhatsApp us on 08088919888.

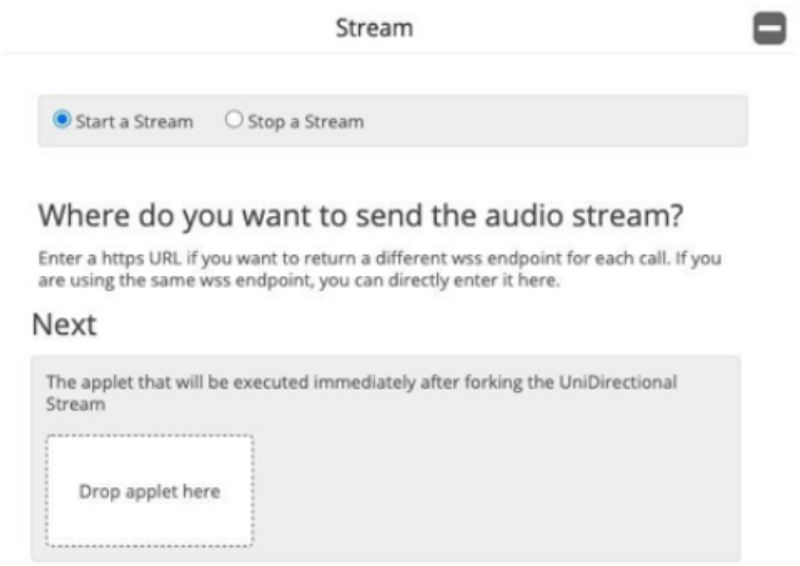
# 2.3.3. Stream Applet

**Unidirectional Streaming**

Unidirectional Streams allow you to Stream live calls over WebSocket in a single direction, from Exotel to the WebSocket Endpoint. Some of the use cases for this are live transcription, realtime monitoring of agents, realtime coaching etc.

**Configuring a Unidirectional Stream-Stream applet**

You can enable unidirectional streaming on a call flow using the Streaming Applet.




**The applet takes 3 parameters :**

**1. Action** - You can either start a new stream or stop a stream that you started earlier in the same call flow. You will use the Stop action if you have started a unidirectional Stream earlier in the same flow. When you choose stop, that is the only input you need to configure

**2. URL** - This is the URL to which Exotel will stream the voice media. You can either specify a WSS endpoint or an HTTPS endpoint. If you specify an HTTP/HTTPS endpoint, Exotel expects the HTTPS endpoint to return a WSS URL in its response. This is to allow

a. Dynamic endpoints for the same call flow
b. Have dynamic custom parameters that can be passed to the websocket endpoint to handle any application specific customization

When you specify a https endpoint, it must return a json with the key “url”

JSON

```
{
"url" : "wss://streamhandler.yourdomain.com"
}
```

On receiving this, Exotel will initiate a connection to wss://streamhandler.yourdomain.com

**3. Next Applet**

In the case if Unidirectional Stream, the Stream would be created and the call flow proceeds to the next applet configured

# 2.3.4. Greeting Applet

The Greeting applet plays a recorded voice message to greet your callers. Use it as the first step in your call flow to welcome callers before routing them.

## Features

- **Phone recording** – Record a greeting using your own phone
- **File upload** – Upload a pre-recorded audio file (8000 Hz Mono .wav format)
- **Text-to-speech** – Enter text and let the system generate speech automatically (Robocop)

## Usage

The Greeting applet is typically the first applet in an incoming call flow:

```
Incoming Call → Greeting → IVR Menu → Connect
```

## Configuration

1. In the Exotel Dashboard, open your call flow editor
2. Drag the **Greeting** applet into your flow
3. Choose your greeting method:
    - **Record via phone**: Enter your phone number and Exotel will call you to record
    - **Upload file**: Upload an 8000 Hz Mono .wav file
    - **Text-to-speech**: Type your greeting message

## Best Practices

- Keep greetings short and clear (under 15 seconds)
- If recording calls, use the Greeting applet to inform callers: *"This call may be recorded for quality assurance purposes"*
- Use text-to-speech for quick prototyping, then switch to professional recordings for production

# 2.3.4.1. Greeting using dynamic text or audio from URL

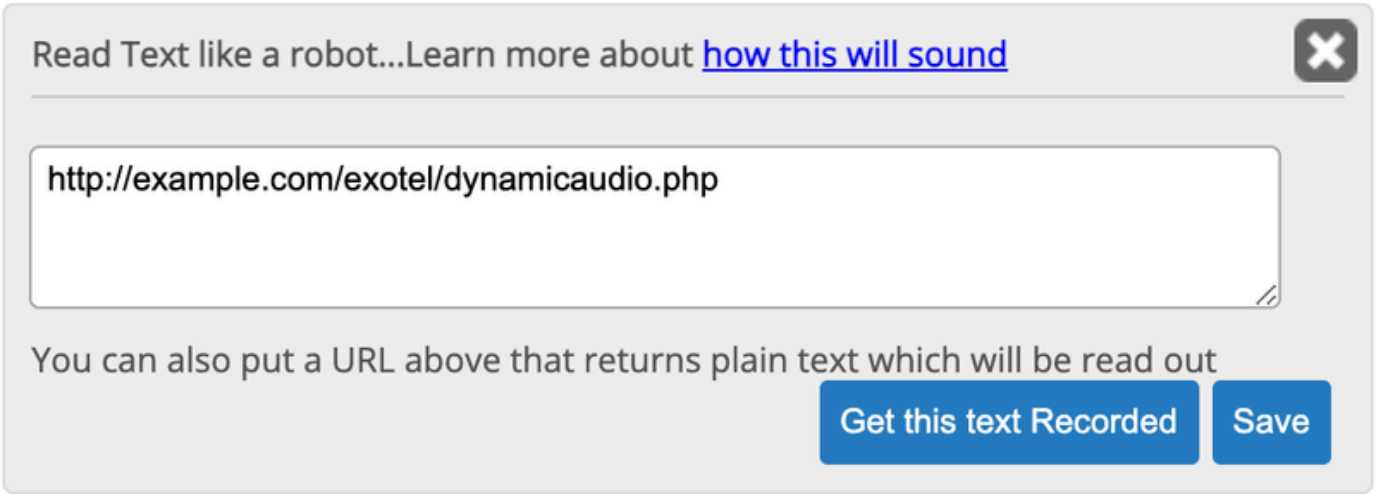
**What is usual?**

Typically, while configuring your app, you would enter the text or upload the audio that you want your callers to listen to. This is static in nature - i.e., all your callers will hear the same message/audio.

**How to make it dynamic?**

In Exotel, you have an option to make this greeting dynamic - i.e., you can potentially play a different greeting to different callers.

In the Greeting (or Menu) applet, you can specify a URL (instead of a static text) in the "Read text like a robot" option. See below.




When the text box contains a URL, Exotel will make an HTTP GET request to that URL (which is hosted at your server). Your server can now return either a text or a URL (to an audio file). If it returns a text, the text will be converted to audio using our Text-To-Speech (TTS) engine and then played out.

The applets mentioned above are a part of the Custom App section also called App Builder, to know more about App Builder please follow----> **Exotel App Builder**

**HTTP Request from Exotel (to your URL)**

The GET request that Exotel makes to the URL will have the following query parameters:

| PARAMETER NAME | VALUE |
|---|---|
| CallSid | string; unique identifier of the call |
| From | string; the number of the calling party |
| To | string; your Exotel company number that is being called; this will be from your "Company Numbers" page |
| DialWhomNumber | string; the number that is being called currently (This might be empty also) |
| CustomField | string; This is set if CustomField was provided while initiating an Outbound call (Not applicable for IB calls) |

To know more about passing custom fields, while making an Outbound call, please follow ---> APIs

**HTTP Response from your web server**

Your web server:

- MUST set the **Content-Type** HTTP header to '**text/plain**' and nothing else.
- MUST support a HEAD request from Exotel and return the exact same headers that it would for a GET request.

In case you want to read out a text,

- The HTTP response body must contain the text to be read out in plain English - and nothing else. Such as this:

```
Thanks for calling Super Kart. Your order has already been shipped. It will reach you in 2 to 3 business days.
```

In case you want to play an audio

- The HTTP response body must contain the URL of the audio file (one per line). Such as this:

```
http://example.com/first-audio.wav
http://example.com/second-audio.wav
```

- The Audio files **MUST** be in the .wav/.mp3, 8Khz Mono format. The bit depth must be 16 bit.

NOTE: All other formats are **not** supported (like 16Kz Mono/Stereo etc)

**Caching of the audio files**

Exotel caches the audio files that it downloads and plays - so that the next time you send the same file, we don't need to download and it will be faster.

Exotel caches the file based on the **name of the URL.** So in case you are changing first-audio.wav, you will need to send in another name the next time (Ex: http://example.com/first-audio-new.wav)

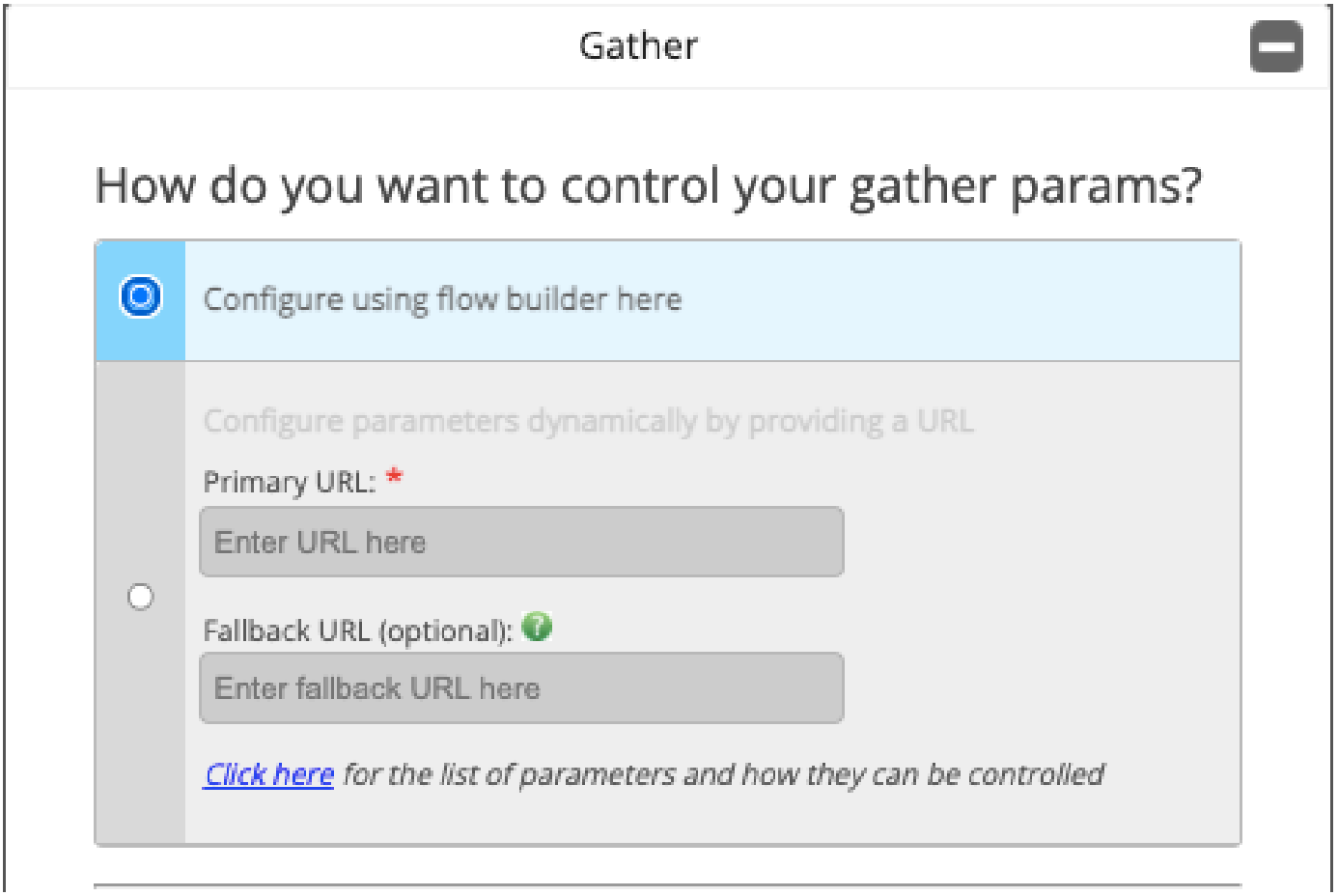
Note: If you do not wish to play the dynamic content, you can also play a voice message by adding the context instead of the URL. Please find the link useful on How can I request a voice over and where can I find my voice over recordings?

# 2.3.5. Gather Applet

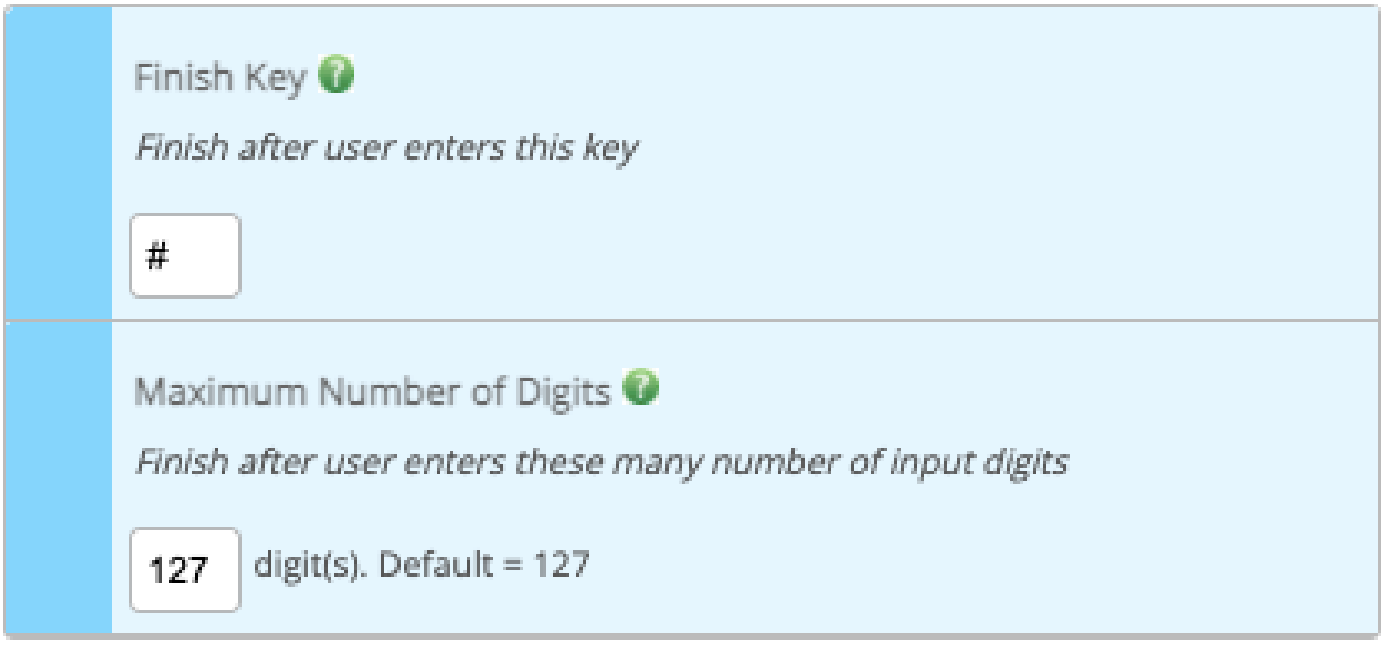
This applet allows you to take numeric information from the user when they are pressing something on their keypads. The information gathered can be used for mobile number input, order id verification etc. along with the Pass-thru applet to process such information in real-time and proceed with an appropriate set of actions with the user on call.

You can choose to configure the gather applet parameters using the flow builder itself or control it dynamically through a URL. However, you'll need to configure the transitions (next applet) while building the flow irrespective.

**1. Configure using flow builder**

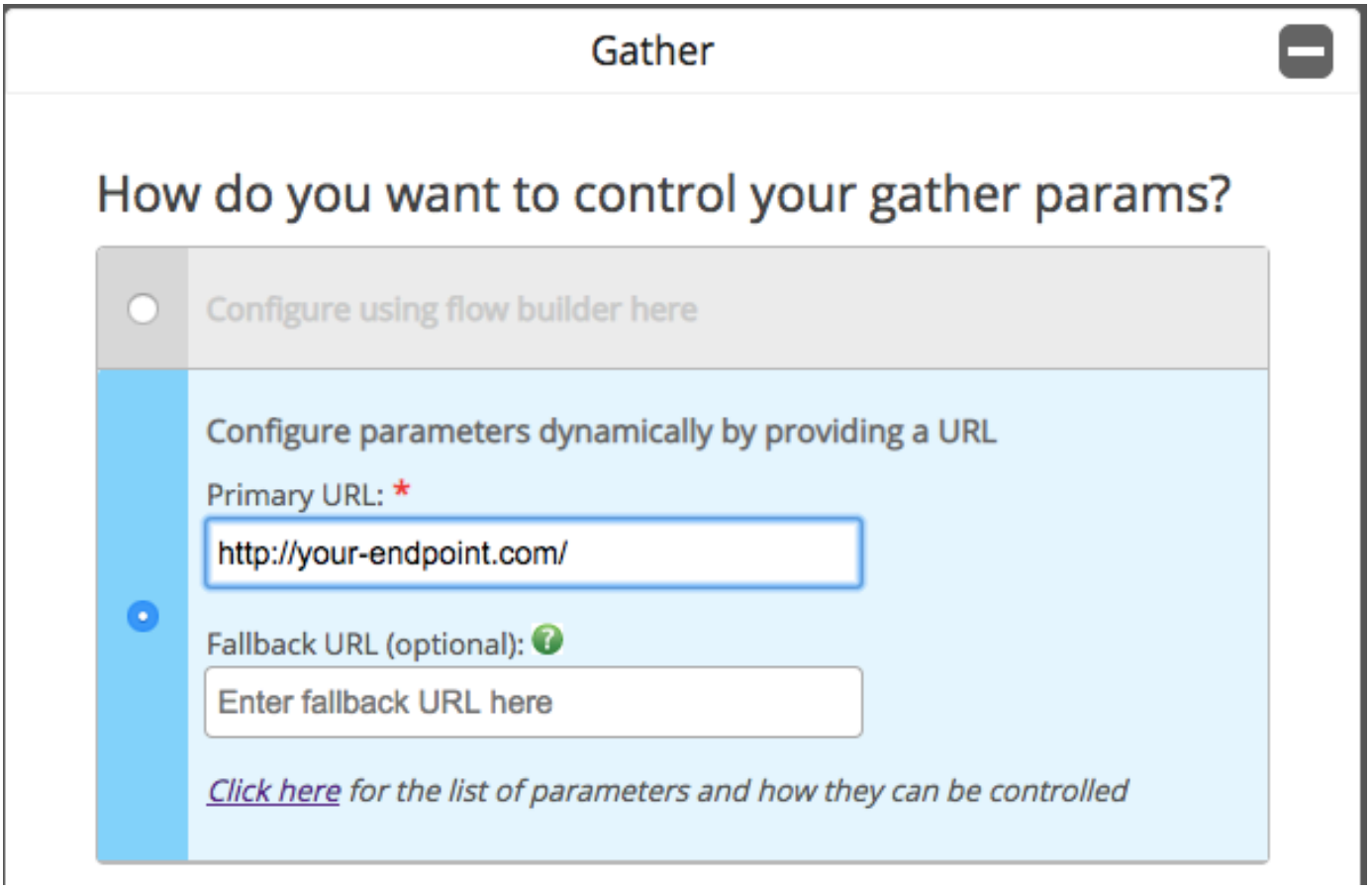



Play a prompt to the caller to enter the input




Specify Key Input behaviour

**2. Configure parameters dynamically by providing a URL**




In case you choose to configure the parameters dynamically using your application URL, you can set a 'Primary URL' for handling the requests. You can also set a 'Fallback URL' (optional) which will be contacted in case something goes wrong with your 'Primary URL'.

**Request**

If an application URL is set for gather applet, Exotel will make a GET request to the URL with the call details as URL-encoded HTTP query parameters.

The following are the parameters of the GET request. Note that only some of this list may be passed to your application - depending on what stage of the flow you have placed the gather applet.

Header:

| | |
|---|---|
| Exotel-Version | This value will indicate the version of gather applet parameters against which your endpoint's response will be validated.<br><br>Current Version: 1.0 |

Query Parameters:

| Parameter Name | Description |
|---|---|
| CallSid | Unique identifier of the call |
| CallFrom | In case of an outgoing call, it'll be set to the number from which the call is made. In case of an incoming call, it'll be set to the number from which the call is received |
| CallTo | In case of an outgoing call, it'll be set to the number being dialled out. In case of an incoming call, it'll be set to the number where the call landed. |
| Call status | The status of the call depends on what stage it is at.<br><br>Possible values: 'queued', 'ringing', 'in-progress', 'completed', 'busy', 'failed', 'no-answer', 'canceled' |
| Direction | The direction of the call. Possible values: 'incoming' or 'outbound-dial' |
| Created | Timestamp when the call is created |
| DialCallDuration | Value in seconds from the time call is triggered to the second leg of the call till it is over (including conversation time). This value can be set to zero depending on the previous applet and if there's no second leg in the call flow. |
| StartTime | Timestamp when the call is started |
| EndTime | 1970-01-01 05:30:00 // Unix time (also known as POSIX time or epoch time)<br><br>Note that this would be a constant value and you may instead trigger our Call details API a few minutes after the call has been completed, to get the accurate information |
| CallType | Scenario<br>Value<br>IVR only, no connect applet<br>call-attempt<br>Call conversation happened<br>completed<br>The client hung up during connect applet<br>client-hangup<br>Connect applet, no agent picked up<br>incomplete<br>Went to voicemail applet<br>voicemail |
| DialWhomNumber | Shows the number of the agent who was dialled to last |
| flow_id | Flow id associated with the call |
| From | In case of an incoming call, it is the number of the caller. In the case of an outgoing call, it is the number of the first leg of the call. |
| To | In case of an incoming call, it is the ExoPhone into which the call came. In the case of an outgoing call, it is the number to which the call was made. |

| Parameter Name | Description |
|---|---|
| CurrentTime | Current server time (format : yyyy-mm-dd, hh:mm:ss) |

*These parameters will be passed if certain conditions are met as described below:

| Parameter Name | Description |
|---|---|
| DialCallStatus | This will denote what happened with the second leg of the call if the previous applet was “connected”.<br><br>Possible values: 'completed', 'busy', 'no-answer', 'failed', 'canceled' |
| Legs | An array that will denote detailed information about each leg attempt if the previous applet was “connect”.<br><br>Sample:<br>Legs[0][CauseCode]=NORMAL_CLEARING<br>Legs[0][Cause]=16<br>Legs[0][Type]=single<br>Legs[0][OnCallDuration]=21<br>Legs[0][Number]=07200498123<br><br>Meaning:<br>• Legs[i]: I denotes the index of the attempt in the connect applet. If there are multiple numbers attempted, the array’s length will be equal to the total attempts.<br>• CauseCode: Cause code of the call as returned by the operators<br>• Code: Numeric representation of the cause code<br>• Type: single or parallel<br>• OnCallDuration: Time spent by the leg on call. This value could be 0 if there was no conversation with the attempted leg.<br>• Number: Phone number of the attempted leg |
| digits | ‘digits’ will be passed if there was a 'Gather' or ‘IVR’ applet before this applet and will be equal to the input digits that were entered. NOTE: This parameter comes with a double quote (") before and after the number. You'll have to trim() this parameter for double quotes (") to get the actual digits. |
| CustomField | If the call was initiated via API, the value that was passed in CustomField in the API call will be set here. |
| RecordingUrl | This will be populated if the previous applet was "voicemail". It will contain the URL of the voicemail recording. There could be a delay before the recording can be accessed depending on the length of the recording file. |

**Response**

This is the response Exotel will expect to the GET request from your Gather Application URL. The response will decide what parameters will be set while executing gather during the call flow.

Response Header:

| | |
|---|---|
| Content-Type | application/json |

Sample:

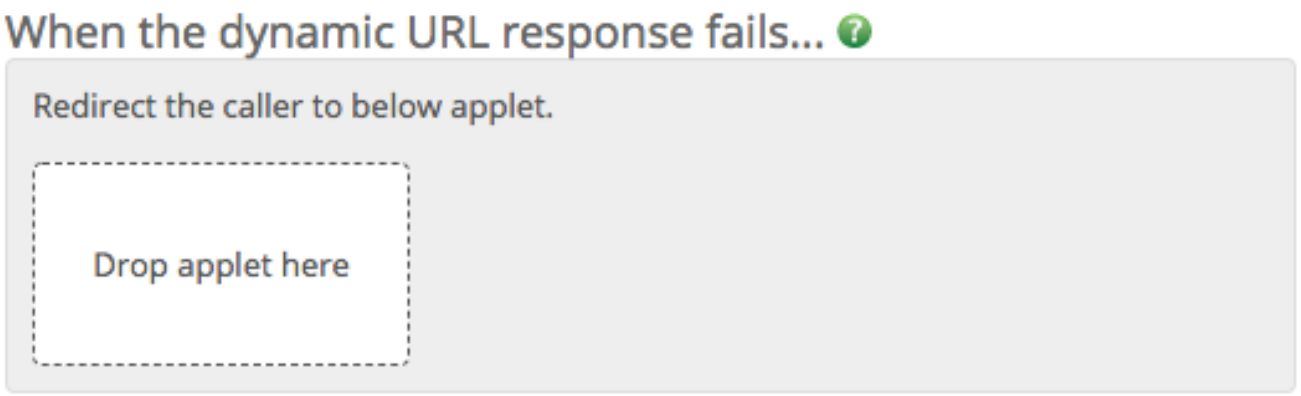
```
{"gather_prompt": {    "text": "Hello Kovalan, please provide your order id",  },"max_input_digits":
5,"finish_on_key": "#","input_timeout": 6,"repeat_menu": 2,"repeat_gather_prompt": {    "text": "It seems that
you have not provided any input, please try again."  }}
```

Explanation:

| Parameter | Type | Behaviour | Description |
|---|---|---|---|
| gather_prompt | string | Mandatory | URL of the audio file or text which will be played out<br><br>Possible Values:<br><br>{<br>"audio_url": "http://your-endpoint.com/test-audio.mp3"<br>}<br>OR<br>{<br>"text": "Please enter your mobile number"<br>} |
| max_input_digits | integer | Optional;<br><br>default = 255 | A maximum number of digits which are expected by the user to be entered, after which the gather should end successfully. |
| finish_on_key | string | Optional;<br><br>default = # | Input key after which the gather should end successfully.<br><br>Allowed: '' (empty string), 0-9, *, #<br><br>Empty would mean there is no finish key.<br><br>If null or not set, it'll take the default value #. |
| input_timeout | integer | Optional;<br><br>default = 5 seconds | Time period in seconds between each key press within which the user has to enter another input key (first input included) |
| repeat_menu | integer | Optional;<br><br>default = 0 | Number of times menu has to be repeated if there is no input provided by the user |
| repeat_gather_prompt | string | Optional;<br><br>default = gather_prompt | URL or text to be played out if the menu is repeated i.e. in case repeat_menu > 0<br><br>Possible values:<br>{<br>"audio_url": "http://your-endpoint.com/test-audio.mp3"<br>}<br>OR<br>{<br>"text": "Please enter your mobile number"<br>} |

*Above set of parameters can also be controlled through the dashboard if one opts to configure the gather applet using the flow builder instead of the Application URL.

**NOTE:**




When mandatory params are missing, URL is unresponsive (non-2xx) or type validation fails from both Primary and Fallback URL (if set), it will go to URL failure case.

- If optional parameters are not passed in response, it will be set to default.
- If both finish_on_key and max_input_digits are passed, whichever criteria are met first will occur i.e. if finish_on_key is pressed or max_input_digits is reached as per the user's input

Cases, where Fallback URL will be triggered, are:

- Primary URL does not return HTTP 200 OK
- Primary URL doesn't return within the timeout period (5 seconds)
- Primary URL returns invalid response:
  - Content-Type should be application/json
  - Mandatory parameters are present
  - audio_url / text should be http/https and returns 200 HTTP code

# 2.3.6. Passthru Applet

The Passthru applet enables dynamic call flow control by communicating with your application in real-time. When a call reaches this applet, Exotel sends call details to your application URL and uses your response to determine the next step.

## How It Works

1. A call reaches the Passthru applet in your flow
2. Exotel sends an HTTP request to your Application URL with incoming call details
3. Your application processes the data (e.g., looks up the caller in a CRM)
4. Your application responds with an HTTP status code
5. The call flow branches based on your response

## Decision Making

The Passthru applet supports **binary decisions** using HTTP status codes:

| Response Code | Action |
|---|---|
| 200 OK | Route to **Choice A** (the applet connected to the "200" branch) |
| 302 Found | Route to **Choice B** (the applet connected to the "302" branch) |

## Parameters Sent to Your URL

When a call reaches the Passthru applet, Exotel sends details about the incoming call to your Application URL. For the complete list of parameters, refer to the Passthru applet documentation.

## Use Cases

- **VIP routing** – Check if the caller is a premium customer and route to priority support
- **Business hours** – Check if it's within office hours and route accordingly
- **CRM integration** – Look up the caller and route to their assigned account manager
- **Load balancing** – Distribute calls based on agent availability

## Example Flow

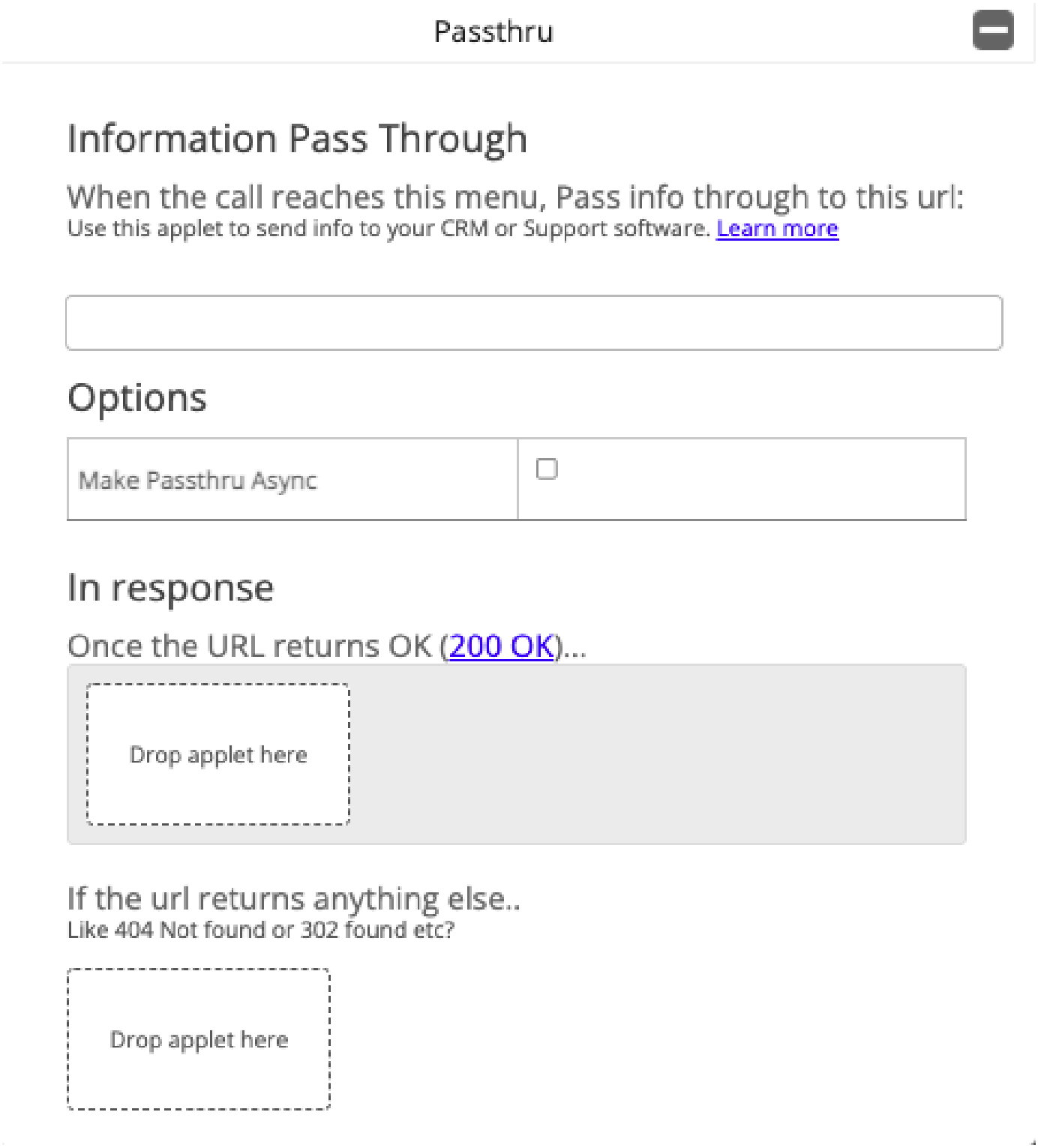
```
Incoming Call → Passthru (check CRM)
                ├── 200 OK → Connect (to assigned agent)
                └── 302 Found → IVR Menu (general support)
```

**tip**

Use the Passthru applet instead of StatusCallbackEvents when you're using VoiceUrl for call flows.

# 2.3.6.1. Working with Passthru Applet

Using the Passthru applet in the call flow you can get Exotel to talk to your Application URL and pass on details about the call as and when it happens. Your application can now process this information and decide which path (success/failure) the flow should take next. This applet helps to dynamically control the flow as well as store the call details in your CRM system or other applications.



The Passthru applet passes the call details to the URL configured. It makes a GET request to the URL with the call details as URL-encoded HTTP query parameters.

Passthru can be used in 2 modes (by toggling the checkbox to 'Make Passthru Async'):

- Sync: Exotel will immediately pass the call details synchronously during the call flow execution. In this case, it is possible only to make a binary decision with Passthru. For example, You can return back a "`200 OK`" for choice A and "`302 Found`" for Choice B. Read more about HTTP response codes here. Please note, that the person on call will have to wait for the next action to be executed until your URL responds back.
- Async: Exotel will asynchronously pass the call details without interrupting the call flow execution. Use this option if you don't want to dynamically change flow execution i.e. select the next applet based on Passthru response. This will ensure the person on call doesn't wait for the time period it takes for Passthru to be executed i.e. your URL to respond back.

The following are the query parameters of the GET request. Note that only some of this list may be passed to your application - depending on what stage of the flow you have placed the pass-through applet.

**Query Parameters:**

| Parameter Name | Description |
|---|---|
| CallSid | Unique identifier of the call |
| CallFrom | In case of an outgoing call, it'll be set to the number from which the call is made. In case of an incoming call, it'll be set to the number from which the call is received |
| CallTo | In case of an outgoing call, it'll be set to the number being dialed out. In case of an incoming call, it'll be set to the number where the call landed. |
| CallStatus | The status of the call depends on what stage it is at. Possible values: 'queued', 'ringing', 'in-progress', 'completed', 'busy', 'failed', 'no-answer', 'canceled' |
| Direction | The direction of the call. Possible values: 'incoming' or 'outbound dial |
| Created | Timestamp when the call is created |
| DialCallDuration | Value in seconds from the time the call is triggered to the second leg of the call till it is over (including conversation time). This value can be set to zero depending on the previous applet and if there's no second leg in the call flow. |
| StartTime | Timestamp when the call is started |
| EndTime | 1970-01-01 05:30:00 // Unix time (also known as POSIX time or epoch time)<br>Note that this would be a constant value and you may instead trigger our Call details API a few minutes after the call has been completed to get accurate information |
| CallType | <br><br>Scenario<br>Value<br>IVR only, no connect applet<br>call-attempt<br>Call conversation happened<br>completed<br>The client hung up during connect applet<br>client-hangup<br>Connect applet, no agent picked up<br>incomplete<br>Went to voicemail applet<br>voicemail |
| DialWhomNumber | Displays the number of the agent who was dialed to last |
| From | In case of an incoming call, it is the number of the caller. In the case of an outgoing call, it is the number of the first leg of the call. |
| To | In case of an incoming call, it is the ExoPhone on which the call landed. In case of an outgoing call, it is the number to which the call was made. |
| CurrentTime | Current server time (format : yyyy-mm-dd hh:mm:ss) |

*These parameters will be passed if certain conditions are met as described below:

| Parameter Name | Description |
|---|---|
| DialCallStatus | This will denote what happened with the second leg of the call if the previous applet was “connect”.<br>Possible values: 'completed', 'busy', 'no-answer', 'failed', 'canceled' |
| Legs | An array that will denote detailed information about each leg attempt if the previous applet was “connect”.<br><br>Sample:<br>Legs[0][CauseCode]=NORMAL_CLEARING<br>Legs[0][Cause]=16<br>Legs[0][Type]=single<br>Legs[0][OnCallDuration]=21<br>Legs[0][Number]=07200498123<br><br>Meaning:<br>• Legs[i]: i denotes the index of the attempt in the Connect applet. If there are multiple numbers attempted, the array’s length will be equal to the total attempts.<br>• CauseCode: Cause code enum of the call as derived from the ISDN cause code returned by the carriers. A list of CauseCode can be found here<br>• Cause: Numeric representation of the CauseCode. A list of Causes can be found here<br>• Type: single or parallel<br>• OnCallDuration: Time spent by the leg on call. This value could be 0 if there was no conversation with the attempted leg.<br>• Number: Phone number of the attempted leg |
| digits | ‘digits’ will be passed if there was a 'Gather' or ‘IVR’ applet before this applet and will be equal to the input digits that were entered. NOTE: This parameter comes with a double quote (") before and after the number. You'll have to trim() this parameter for double quotes (") to get the actual digits. |
| CustomField | If the call was initiated via API, the value that was passed in CustomField in the API call will be set here. |
| RecordingUrl | This will be populated if the previous applet was "Connect" or "Voicemail". It will contain the URL of the conversation in the Connect Applet (if enabled) or voicemail recording. There could be a delay before the recording can be accessed depending on the length of the recording file. |
| OutgoingPhoneNumber | This will be populated if the previous applet was “connect”. It indicates which ExoPhone (Virtual Number) was used to dial out in the Connect Applet. |

Deprecated parameters:

The following parameters have been deprecated and are no longer supported. Some of these values might be passed inadvertently and applications should not rely on them anymore

| |
|---|
| RecordingAvailableBy |
| ForwardedFrom |
| ProcessStatus |
| tenant_id |
| flow_id |

**Samples Responses for different scenarios.**

1) Sample Response for a Completed call where the previous applet is a Connect Applet with Recording URL:

| | |
|---|---|
| CallSid | 56ad12c0e29d[redacted] |
| CallFrom | 086605[redacted] |
| CallTo | 080468[redacted] |
| CallStatus | completed |
| Direction | incoming |
| Created | Wed, 27 Dec 2023 12:17:05 |
| DialCallDuration | 19 |
| RecordingUrl | https:[redacted]/E... |
| StartTime | 2023-12-27 12:17:05 |
| EndTime | 1970-01-01 05:30:00 |
| DialCallStatus | completed |
| CallType | completed |
| DialWhomNumber | 094490[redacted] |
| flow_id | 7[redacted] |
| tenant_id | 15 |
| From | 086605[redacted] |
| To | 080468[redacted] |
| RecordingAvailableBy | Wed, 27 Dec 2023 12:22:26 |
| CurrentTime | 2023-12-27 12:17:26 |
| OutgoingPhoneNumber | 080468[redacted] |
| Legs | [{"Number":"094490[redacted]","Type":"single","OnCallDuration":"8","CallerId":"080468[redacted]","CauseCode":"NORMAL_CLEARING","Cause":"16","DisconnectedBy":"Caller"}] |

2) Sample Response for a Completed call where previous applet is a Gather/IVR Applet:

| | |
|---|---|
| CallSid | b8013b5546750a |
| CallFrom | 086605 |
| CallTo | 080468 |
| CallStatus | in-progress |
| Direction | incoming |
| Created | Wed, 27 Dec 2023 12:26:25 |
| DialCallDuration | 25 |
| RecordingUrl | https:/ /E... |
| StartTime | 2023-12-27 12:26:25 |
| EndTime | 1970-01-01 05:30:00 |
| DialCallStatus | completed |
| CallType | completed |
| DialWhomNumber | 094490 |
| ProcessStatus | ready |
| flow_id | 7 |
| tenant_id | 15 |
| From | 086605 |
| To | 080468 |
| RecordingAvailableBy | Wed, 27 Dec 2023 12:32:01 |
| CurrentTime | 2023-12-27 12:27:01 |
| digits | "15246" |

3) Sample Response for a Completed call where the call was initiated via an API, with the value that was passed in CustomField of the API request.

| | |
|---|---|
| CallSid | 402cd0139019[redacted] |
| CallFrom | 086605[redacted] |
| CallTo | 080468[redacted] |
| CallStatus | completed |
| Direction | outbound-dial |
| Created | Wed, 27 Dec 2023 12:53:35 |
| DialCallDuration | 35 |
| RecordingUrl | https:[redacted]/E... |
| StartTime | 2023-12-27 12:53:35 |
| EndTime | 1970-01-01 05:30:00 |
| DialCallStatus | completed |
| CallType | completed |
| DialWhomNumber | 094490[redacted] |
| flow_id | 7[redacted] |
| tenant_id | 15 |
| From | 086605[redacted] |
| To | 080468[redacted] |
| RecordingAvailableBy | Wed, 27 Dec 2023 12:59:13 |
| CurrentTime | 2023-12-27 12:54:13 |
| CustomField | Hello there |
| OutgoingPhoneNumber | 080468[redacted] |
| Legs | [{"Number":"09449[redacted]","Type":"single","OnCallDuration":... |

**Note:** The response you get from Passthru cannot be customised. However, you can add a Custom field only via the **Outgoing call to connect number to a call flow API**.

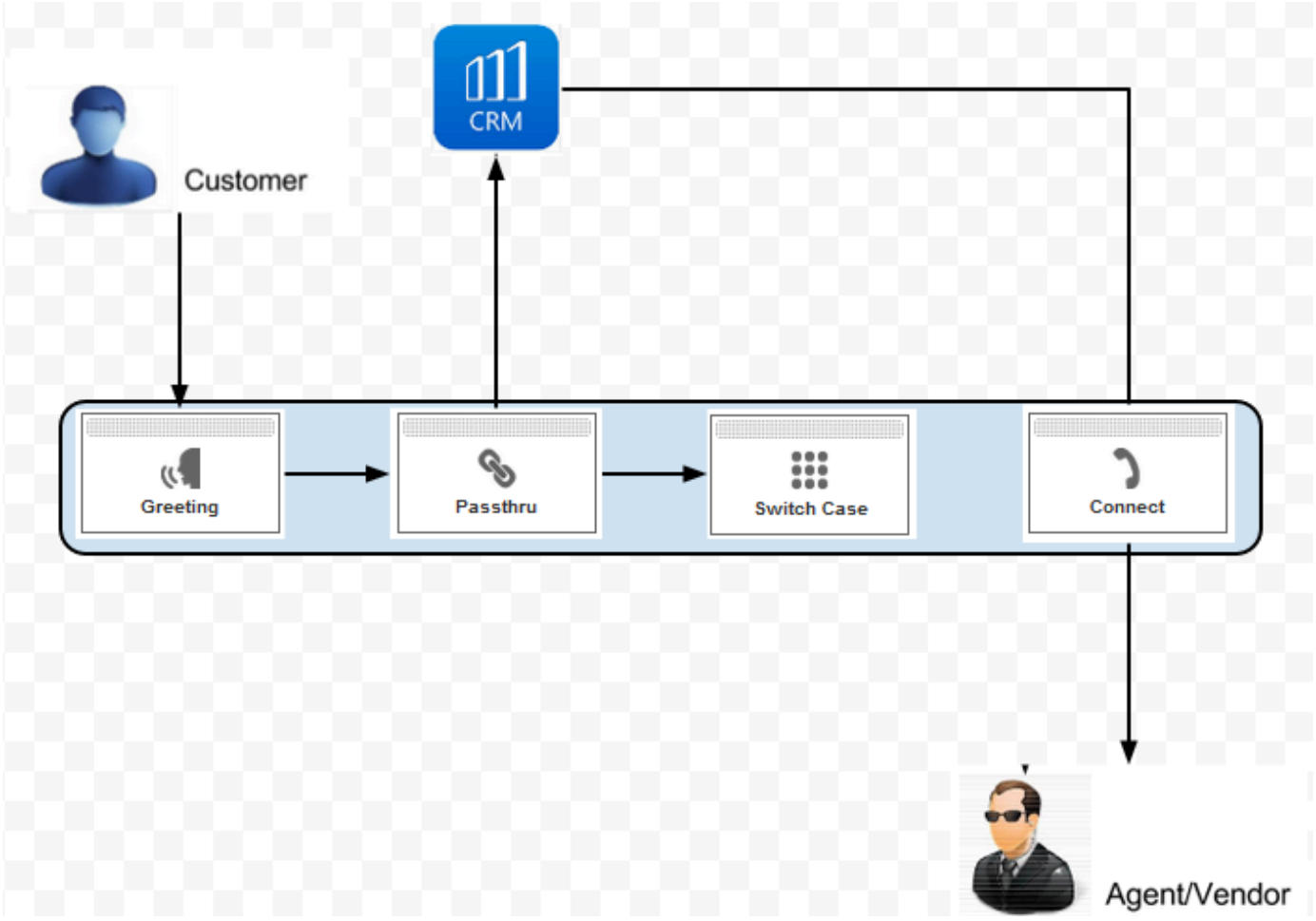
Looking to streaming metadata using Passthru? Learn how to receive stream metadata post-call, link **Passthru Applet**

# 2.3.7. Switcase Applet

This applet helps customers to streamline their flows based on multiple responses. It helps them to create different workflows based on different responses given.

Ex: For a condition-based routing(Geo routing API) or defining complex call flow sceanrios based on previous applets

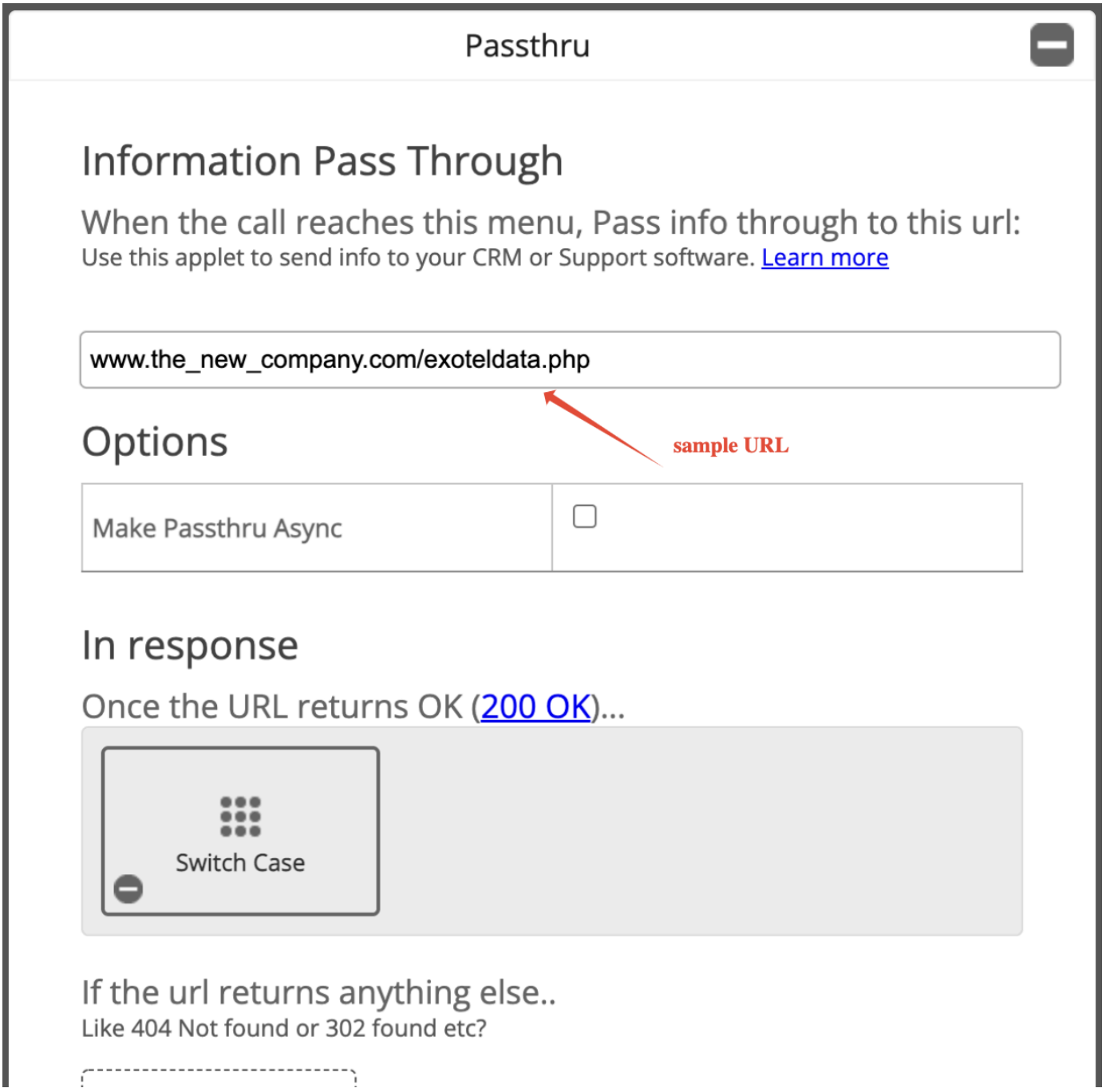
The call flow would be like this:




The steps followed here are:

1. Customer Calls a central number

2. The pass-through applet passes the call details to the URL configured

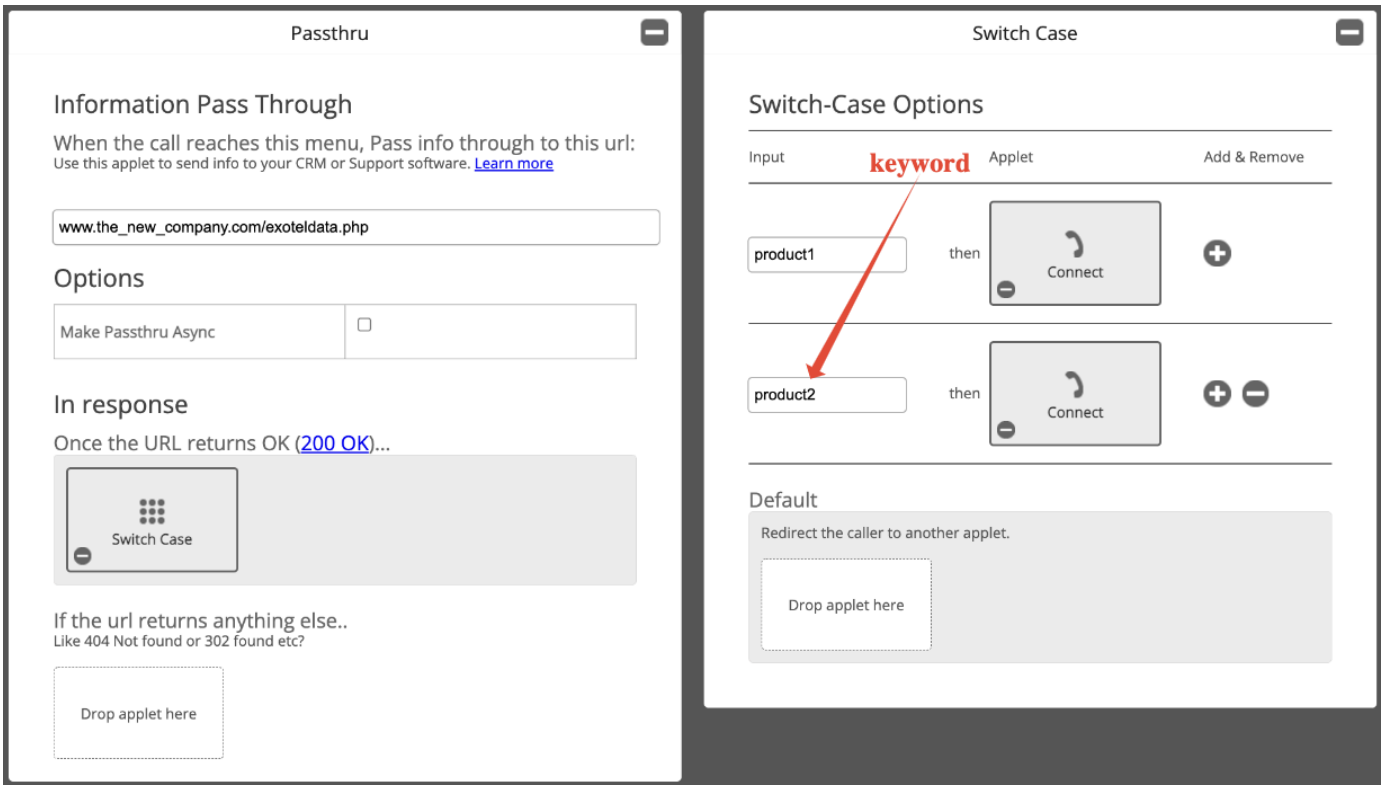



It makes a `GET` request to the URL with the call details as URL-encoded HTTP query parameters.

```
eg:-  www.the_new_company.com/exoteldata.php?
CallSid℗℗=1321360773 &From℗℗=09880847047&To℗℗=08088919888&Direction=incoming&DialCallDuration=7&StartTime=2011-
11-15+18%3A09%3A41&EndTime=0000-00-00+00%3A00%3A00&CallType=call&DialWhomNumber℗℗=09052161119&digits=%2289%22
HTTP/1.1
```

Once Passthru sends the data, the customer needs to read the digits parameter along with the Callsid and return a 200 Ok response along with a keyword in the body of the response (The response needs to be in **`text/plain`** format).

This list of keywords that are returned in the passthru applet will be checked in the switch case applet, where these keywords are pre-defined and are set to do corresponding actions (This could be connecting to a group (Connect applet), playing a greeting (greeting applet) and others). Please find one such example below.




Example:

CRM now needs to send a keyword to be read by SWITCH CASE, which would connect the call to a specific team.

The CRM needs to return response like below in text/plain format

```
{"select":"<Name/option in the switchecase>"}
{"select":"product1"}
```

This option is read by the switch case applet and tries to follow the next part of the call flow which is "connect" to reach the appropriate group.

This can be used for Dynamic routing in case of marketplace scenarios.

If you have any questions or concerns, please connect with us using the chat widget on your Exotel Dashboard or Whatsapp us on 08088919888

# 2.3.8. Transfer Applet

The Transfer applet lets you build modular call flows by transferring between multiple flows. Instead of creating one large, complex flow, you can break it into smaller reusable flows and use the Transfer applet to connect them.

## How It Works

1. Create separate call flows for different functions (e.g., Voicemail flow, Support flow)
2. In your main flow, use the Transfer applet to jump to another flow
3. The call continues in the target flow

## Use Case: Reusable Voicemail

Instead of duplicating voicemail configuration in every branch of a complex IVR:

**Without Transfer** (repetitive):

```
IVR Menu → Press 1 → Connect → (no answer) → Voicemail setup
         → Press 2 → Connect → (no answer) → Voicemail setup (duplicate)
         → Press 3 → Connect → (no answer) → Voicemail setup (duplicate)
```

**With Transfer** (modular):

```
Voicemail Flow: Greeting → Voicemail

IVR Menu → Press 1 → Connect → (no answer) → Transfer to Voicemail Flow
         → Press 2 → Connect → (no answer) → Transfer to Voicemail Flow
         → Press 3 → Connect → (no answer) → Transfer to Voicemail Flow
```

## Configuration

1. Create the target flow (e.g., a "Voicemail" flow) in the Exotel Dashboard
2. In your main flow, drag the **Transfer** applet
3. Select the target flow from the dropdown

## Best Practices

- Use Transfer for any functionality that repeats across multiple flows
- Keep individual flows focused on a single responsibility
- Name your flows clearly so they're easy to reference

# 2.3.9. IVR Menu Applet

The IVR (Interactive Voice Response) Menu applet presents callers with a voice menu and routes them based on their DTMF (keypad) input.

## How It Works

1. The applet plays a recorded prompt (e.g., "Press 1 for Sales, Press 2 for Support")
2. The caller presses a key on their phone
3. The call is routed to the applet or flow connected to that key

## Configuration

1. In the Exotel Dashboard, open your call flow editor
2. Drag the **IVR Menu** applet into your flow
3. Configure the voice prompt (record, upload, or text-to-speech)
4. Map each key (0-9, *, #) to the next applet in the flow

## Example Flow

```
Incoming Call → Greeting ("Welcome to Acme Corp")
             → IVR Menu ("Press 1 for Sales, 2 for Support, 3 for Billing")
                ├── 1 → Connect (Sales team)
                ├── 2 → Connect (Support team)
                ├── 3 → Connect (Billing team)
                └── No input / Invalid → Repeat or Hangup
```

## Best Practices

- Keep menus simple – no more than 4-5 options
- Always provide a fallback for invalid input or no input
- Put the most common option first
- Use the Greeting applet before the IVR Menu to welcome callers

# 2.3.10. Voicemail Applet

The Voicemail applet allows callers to leave a recorded voice message when no one is available to take the call. Messages are routed to specified groups or users.

## How It Works

1. A call reaches the Voicemail applet (typically after a failed Connect attempt)
2. The caller hears a prompt to leave a message
3. The caller records their message and hangs up
4. The voicemail is stored and routed to the configured recipients

## Features

- **Route to groups or users** – Send voicemail notifications to specific teams or individuals
- **Tracking** – All voicemails are traced and tracked in the system
- **Integration** – Combine with the Email applet to send voicemail notifications

## Configuration

1. In the Exotel Dashboard, open your call flow editor
2. Drag the **Voicemail** applet into your flow
3. Configure the recipient group or user
4. Optionally add a Greeting applet before it with a custom prompt

## Example Flow

```
Incoming Call → Connect (to support team)
             → (no answer) → Greeting ("Please leave a message after the beep")
             → Voicemail
```

# 2.3.11. Hangup Applet

The Hangup applet terminates a call. Use it as the final step in a call flow to cleanly end the conversation.

## How It Works

When a call reaches the Hangup applet, the call is immediately disconnected.

## Usage

Place the Hangup applet at the end of any branch in your call flow where you want the call to terminate:

```
Incoming Call → Greeting → IVR Menu
                          ├── 1 → Connect → Hangup
                          ├── 2 → Connect → Hangup
                          └── No input → Greeting ("Goodbye") → Hangup
```

## When to Use

- At the end of every call flow branch to ensure calls are properly terminated
- After a Greeting applet that plays a final message
- As a fallback for unhandled IVR inputs

# 2.3.12. SMS Applet

The SMS applet sends an SMS message during a call flow. Use it to send confirmation messages, follow-up information, or notifications triggered by call events.

## Modes

### Static Mode

Send a fixed, pre-configured message to the caller.

- Configure the message text directly in the applet
- The same message is sent every time

### Dynamic Mode

Fetch the message content from your application URL.

- The applet sends a GET request to your configured URL
- Your application returns the SMS content as plain text
- Enables personalized messages based on caller data or call context

## Configuration

1. In the Exotel Dashboard, open your call flow editor
2. Drag the **SMS** applet into your flow
3. Choose the mode:
    - **Static**: Enter the message text
    - **Dynamic**: Provide your application URL

## Example Flows

**Post-call confirmation:**

```
Incoming Call → Connect (to agent) → SMS ("Thank you for calling. Your ticket #12345 has been created.")
```

**Dynamic follow-up:**

```
Incoming Call → Passthru (look up order) → SMS (fetch order status from URL) → Hangup
```

## Dynamic Mode Response Format

Your application URL should return the SMS content as **plain text** in the response body. The applet sends a GET request and expects a simple text response.

# 2.3.13. Email Applet

The Email applet sends an email notification with call details when a call reaches this point in the flow. Use it to alert your team about incoming calls, missed calls, or voicemails.

## What Gets Emailed

The email includes details about the call:

- **Caller information** – The caller's phone number
- **Timestamps** – When the call was received
- **Agent details** – Which agent handled the call (if applicable)
- **Call outcome** – Whether the call was answered, missed, or went to voicemail
- **Voicemail status** – Whether a voicemail was left

## Configuration

1. In the Exotel Dashboard, open your call flow editor
2. Drag the **Email** applet into your flow
3. Enter the recipient email address(es)

## Example Flows

**Missed call notification:**

```
Incoming Call → Connect (to support)
             → (no answer) → Email (notify manager) → Voicemail
```

**Call summary:**

```
Incoming Call → Connect (to sales) → Email (send call details to CRM inbox)
```

## Best Practices

- Use Email applets after Connect failures to ensure missed calls are tracked
- Combine with Voicemail for a complete missed-call workflow
- Send emails to shared inboxes or ticketing systems for better tracking

# 3. ExoML

## 3.1. Authentication & Base URL

### Introduction

**ExoML** provides developers with APIs that give low-level control over call legs, media, and bridges.

These APIs are designed for advanced voice orchestration, real-time call control, and custom contact center or voice workflows.

### Base URL

All ExoML Aare served from the Exotel CPaaS API host.

**Base URL format**

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}
```

**Example**

```
https://cpaas-api.in.exotel.com/v2/accounts/ameyo5m
```

**Important**

- ameyo5m is an example Account SID
- Always use {account_sid} as a placeholder

### How to get your API credentials

To use the ExoML, developers must generate API credentials from the Exotel dashboard.

**Steps**

1. Log in to **https://my.exotel.com**
2. Go to **Developer Settings → API Settings**
3. Copy:
    - Account SID
    - API Key
    - API Token

These credentials are required for all Call Control API requests.

### Authentication

All ExoML use **HTTP Basic Authentication**.

- **Username:** API Key

- **Password:** API Token

#### Example

```
curl -u "<API_KEY>:<API_TOKEN>" \
https://cpaas-api.in.exotel.com/v2/accounts/<account_sid>/legs
```

#### Notes

- Credentials are not embedded in the URL
- Credentials are not passed in request bodies
- Every request must include Basic Authentication

## Request Format

#### Required headers

```
Content-Type: application/json
Authorization: Basic <base64(API_KEY:API_TOKEN)>
```

#### URL structure

```
/v2/accounts/{account_sid}/{resource}
```

#### Common resources

- /legs
- /legs/{leg_sid}
- /legs/{leg_sid}/actions
- /bridges
- /bridges/{bridge_sid}
- /bridges/{bridge_sid}/actions

## Example: Create a Leg

#### Endpoint

```
POST /legs
```

#### Full URL

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs
```

#### Example request

```
curl -u "<API_KEY>:<API_TOKEN>" \
-H "Content-Type: application/json" \
-X POST \
https://cpaas-api.in.exotel.com/v2/accounts/<account_sid>/legs \
-d '{
  "contact_uri": "09163816621",
  "exophone": "08030752400",
```

```
  "leg_event_endpoint": "grpc://127.0.0.1:9001"
}'
```

## Next Steps

Once authentication is set up, you can:

- Create and manage call legs
- Bridge multiple legs
- Control media (play, say, record, stream)
- Receive real-time call events via gRPC

Proceed to the **ExoML Introduction** section to explore individual endpoints.

# 3.2. Implementing Your gRPC Endpoint for Real-Time Call Events

To integrate with **ExoML**, your team will need to implement a gRPC endpoint. Our platform uses a **gRPC-based, event-driven architecture**, which means we will push real-time call events directly to your service, offering a more efficient solution than traditional polling.

The service contract, which defines the required methods, is in the **.proto file**.

**Configuration Steps:**

1. **Implement the gRPC Service:**
    - Use the attached .proto schema as your service definition.
    - Your implementation must include the TriggerCallbackEvent method. This is the callback our system will invoke to push event data to you.
2. **Deploy Your Service Endpoint:**
    - The endpoint must be publicly reachable and will act as the "listener" for our events.
    - It needs to be secured with TLS (HTTPS on port 443 is standard). Mutual TLS is optional but recommended.
3. **Register Your Endpoint:**
    - Once deployed, please provide us with the public URL of your gRPC endpoint so we can complete the integration from our end.

callbackTriggerEvent.proto

# 3.3. Authenticating gRPC Callback Events from Exotel

When Exotel pushes real-time call events to your gRPC endpoint, you can optionally require authentication so your service only accepts callbacks from Exotel.

This article describes the authentication method supported for ExoML Programmable Voice API gRPC callbacks.

## Overview

Exotel acts as the gRPC client and invokes your TriggerCallbackEvent method to deliver call events.

If authentication is enabled for your account, Exotel includes credentials on each callback request using HTTP Basic Auth sent as gRPC metadata.

Your gRPC server should validate those credentials before processing the event.

When Basic Auth is configured on your account, Exotel sends the following metadata with every TriggerCallbackEvent request:

authorization: Basic <base64(username:password)>

Where:

- username and password are the credentials configured for your account
- The value is standard HTTP Basic Auth encoding: Base64(username + ":" + password)

## Sample: Authentication header sent by Exotel on gRPC callbacks

When Basic Auth is configured on your account, Exotel includes the following gRPC metadata on every TriggerCallbackEvent request:

authorization: Basic <base64(username:password)>

Example

If your configured credentials are:

- Username: exotel_user
- Password: exotel_pass

Then:

1. Exotel forms: exotel_user:exotel_pass
2. Base64-encodes that string
3. Sends:

authorization: Basic ZXhvdGVsX3VzZXI6ZXhvdGVsX3Bhc3M=

Your gRPC server should:

1. Read the authorization metadata from the incoming request
2. Decode the Basic Auth value
3. Validate the username and password
4. Reject the request if credentials are missing or invalid

Note: Custom headers such as x-exotel-token are not supported. Please validate the standard authorization Basic Auth header.

## How to enable authentication

1. Implement your gRPC endpoint using Exotel's shared .proto schema, including TriggerCallbackEvent. See: https://docs.exotel.com/voice-apis/implementing-your-grpc-endpoint-for-real-time-call-events
2. Decide on the username and password (or API key and token) your service will accept.
3. Share those credentials with Exotel Support / your Exotel contact so they can be configured on your account.
4. Ensure your gRPC server validates the authorization metadata on every callback.

Once configured, Exotel automatically attaches Basic Auth credentials to outbound gRPC callback requests for your account.

## Behavior when authentication is not configured

If Basic Auth credentials are not configured on your account:

- Exotel still pushes events to your public gRPC endpoint
- No authorization metadata is sent

In that case, protect your endpoint using:

- TLS (required)
- Optional mutual TLS (mTLS)
- Network allowlisting / firewall rules
- Your own application-level validation

## IP whitelisting

In addition to Basic Auth, we also support IP whitelisting.

Your firewall / security group should allow inbound gRPC/TLS traffic from the following Exotel IP addresses:

- 35.154.174.161
- 13.233.217.51
- 98.130.5.99

Whitelisting these IPs helps ensure that only Exotel can reach your gRPC callback endpoint.

## Security recommendations

- Always run your gRPC endpoint over TLS (port 443 is standard)
- Prefer enabling Basic Auth for callback verification
- Whitelist Exotel callback source IPs
- Consider mTLS for stronger client authentication where possible
- Rotate credentials periodically and coordinate the update with Exotel
- Reject requests that are missing or have invalid authorization metadata when auth is enabled
- Do not log raw credentials or full Authorization headers in plaintext

## Troubleshooting

Events are reaching your server but auth validation fails

- Confirm you are reading gRPC metadata key authorization (lowercase is typical in gRPC)
- Confirm you expect Basic Auth, not a custom token header
- Confirm the username/password match what was shared with Exotel
- Confirm there are no extra spaces or encoding issues in your decoder

No authorization metadata is present

- Basic Auth may not be configured on your Exotel account yet
- Contact Exotel Support to enable callback Basic Auth credentials for your account

# 3.4. ExoML – Introduction

## Key Concepts

| Concept | Description |
| --- | --- |
| Leg | A single call participant |
| Leg SID | Unique identifier for a leg |
| Actions | Commands executed on a leg |
| External Events | Async signals emitted for state changes |

A **Leg** represents a single participant in a call. Every call is composed of one or more legs, which can be controlled independently using **Leg Actions**.

ExoML allows you to:

- Create a new leg (incoming or outgoing)
- Retrieve leg details
- Update leg configuration
- Execute actions such as Play, Say, Dial, Hold, Mute, Record, Stream, etc.

Base URL

```
https://cpaas-api.in.exotel.com/v2/accounts/{AccountSID}
```

## Authentication

Use **Basic Authentication**.

```
Authorization: Basic <BASE64_APIKEY_APITOKEN>
Content-Type: application/json
```

API credentials are available at:
**my.exotel.com → Developer Settings**

## Related APIs

- **POST /legs** – Create a leg
- **GET /legs/{LegSID}** – Get leg details
- **PUT /legs/{LegSID}** – Update leg
- **POST /legs/{LegSID}/actions** – Execute leg actions
- **GET /legs/{LegSID}/external_events** – Retrieve leg events

# 3.4.1. POST – Create a Leg

## Description

Creates a **call leg**, which represents one side of a call (customer, agent, bot, or SIP endpoint).

A leg can be:

- Outbound (Exotel → destination)
- Inbound (answered and controlled via actions)
- PSTN or SIP-based

Once created, the leg can be **answered, played to, recorded, streamed, bridged, or terminated** using Leg Actions.

## Endpoint

**POST**

```
/v2/accounts/{account_sid}/legs
```

## Full URL

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs
```

## Authentication

- **Type:** HTTP Basic Authentication
- **Username:** API Key
- **Password:** API Token

## Request Headers

```
Content-Type: application/json
Authorization: Basic <base64(API_KEY:API_TOKEN)>
```

## Request Body Parameters

### Mandatory Parameters

| Parameter | Type | Description |
|---|---|---|
| contact_uri | string | Destination to dial. Can be a phone number or a SIP URI (sip:user@domain). |
| exophone | string | Exotel number or SIP identity used as the caller ID. |
| leg_event_endpoint | string | gRPC endpoint where leg events are delivered. Must start with grpc://. |

### Optional Parameters

| Parameter | Type | Default | Description |
|---|---|---|---|
| network_type | string | pstn | pstn or voip. Use voip when contact_uri is a SIP URI. |
| custom_param | string | - | Custom value echoed back in events for correlation. |
| timeout | integer (sec) | 30 | Time allowed for the call to be answered. |
| setup_timeout | integer (sec) | 30 | Time allowed for call setup. |
| ring_timeout | integer (sec) | 30 | Time the destination is rung. |
| time_limit | integer (sec) | 14400 | Maximum duration of the leg (default: 4 hours). |
| amd_enable | boolean | false | Enables Answering Machine Detection. |
| async_amd | boolean | false | If true, AMD result is delivered asynchronously as an event. |
| reference_leg_sid | string | - | Used to colocate legs for low-latency bridging. |
| priority | string | normal | Call priority when CPS controls are enabled. |

## Example Request

```
curl -u "<API_KEY>:<API_TOKEN>" \
-H "Content-Type: application/json" \
-X POST \
https://cpaas-api.in.exotel.com/v2/accounts/<account_sid>/legs \
-d '{
  "contact_uri": "09163816621",
  "exophone": "08030752400",
  "leg_event_endpoint": "grpc://127.0.0.1:9001",
  "network_type": "pstn",
  "timeout": 30,
  "time_limit": 600
}'
```

## Success Response

**HTTP 202 – Accepted**


```
{
  "request_id": "811b113c561a4d35bd8520112eb629f7",
  "method": "POST",
  "http_code": 202,
  "response": {
    "error_data": null,
    "data": {
      "leg_sid": "2QYATxQ9t4UKcUwcqJClSa9OEZ400000",
      "account_sid": "ameyo5m",
      "contact_uri": "09163816621",
      "exophone": "08030752400",
      "network_type": "pstn",
      "created_at": "2023-05-31T08:02:32Z"
    }
  }
}
```


## Response Fields

| Field | Description |
|---|---|
| leg_sid | Unique identifier for the created leg. |
| account_sid | Exotel account identifier. |
| contact_uri | Destination that was dialed. |
| exophone | Caller ID used for the leg. |
| network_type | Network used for the call. |
| created_at | Timestamp when the leg was created. |

## Events Create Leg can generate:

| **Event Name** | **data** |
|---|---|
| “leg_connecting” | "state": "connecting" |
| “leg_ringing” | "state": "ringing" |
| “leg_answered” | "state": "connected" (if amd_enable is true ) "answered_by":"BEEP/ HUMAN / MACHINE" |
| “amd_result_success” | (If amd_enable, async_amd both is set to true ) "state": "connected", "answered_by":"BEEP/ HUMAN / MACHINE" |
| “leg_terminated” | "state": "terminal", "terminal_status": "completed/ busy/ no-answer/ failed" |
| “leg_failed_to_create” | "state": "terminal", "terminal_status": "failed" |

## Error Responses

| HTTP Code | Meaning | Description |
|---|---|---|
| 400 | Bad Request | Missing or invalid parameters |
| 401 | Unauthorized | Invalid API Key or Token |
| 403 | Forbidden | Access not allowed for this account |
| 404 | Not Found | Account SID not found |
| 429 | Too Many Requests | Rate limit exceeded |
| 500 | Server Error | Internal error |

## Notes & Best Practices

- Creating a leg is **asynchronous** – wait for events to know when it is answered
- Always store the returned leg_sid
- Use custom_param to correlate calls with your internal systems
- Ensure your leg_event_endpoint is reachable before creating legs
- Use reference_leg_sid when you plan to bridge multiple legs

## What’s Next?

After creating a leg, you can:

- **Answer or hang up** the leg
- **Play or say** audio
- **Record or stream** the call
- **Bridge** the leg with another leg

➡️ Proceed to **Leg Actions** to control the call.

# 3.4.2. GET – Get Leg Details

## Description

Fetches the current details of a **Leg** using its leg_sid. This is useful to:

- Check the leg state (created / ringing / connected / terminal)
- Fetch timestamps (created / start / end)
- Pull additional runtime statuses (bridge / recording / play / gather / hold, etc.) via fields

## Endpoint

**GET**

```
/v2/accounts/{account_sid}/legs/{leg_sid}
```

## Full URL

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}
```

## Authentication

- **Type:** HTTP Basic Authentication
- **Username:** API Key
- **Password:** API Token

## Request Headers

```
Content-Type: application/json
Authorization: Basic <base64(API_KEY:API_TOKEN)>
```

## Path Parameters

| Parameter | Type | Required | Description |
|---|---|---|---|
| account_sid | string | Yes | Exotel Account SID (from Exotel dashboard) |
| leg_sid | string | Yes | Leg SID returned by Create a Leg |

## Query Parameters (Optional)

### fields

Use fields to request additional runtime statuses in the response.

**Type:** comma-separated string

**Example**

```
fields=bridge_status,recording_status,tx_audio_status,rx_audio_status,hold_status,gather_status,play_status
```

| Field | Description |
|---|---|
| bridge_status | Whether the leg is bridged or not |
| recording_status | Whether recording is active |
| tx_audio_status | Transmit audio status (muted/normal) |
| rx_audio_status | Receive audio status (muted/normal) |
| hold_status | Whether leg is on hold |
| gather_status | Whether DTMF gather is active |
| play_status | Whether audio playback is active |

```
Tip: If you don't pass fields, the API returns the standard leg object without the additional_info section.
```

# Example Request

## Basic request

```
curl -u "<API_KEY>:<API_TOKEN>" \
-X GET \
https://cpaas-api.in.exotel.com/v2/accounts/<account_sid>/legs/<leg_sid>
```

## Request with fields

```
curl -u "<API_KEY>:<API_TOKEN>" \
-X GET \
"https://cpaas-api.in.exotel.com/v2/accounts/<account_sid>/legs/<leg_sid>?
fields=bridge_status,recording_status,hold_status,gather_status,play_status"
```

# Success Response

### HTTP 200 – OK


```
{
  "request_id": "ce7b872225b348f09e80115c1ff569a3",
  "method": "GET",
  "http_code": 200,
  "response": {
    "error_data": null,
    "data": {
      "leg_sid": "2QYATxQ9t4UKcUwcqJClSa9OEZ400000",
      "account_sid": "ameyo5m",
      "contact_uri": "09163816621",
      "exophone": "08030752400",
      "caller_id": "8030752501",
      "direction": "outbound",
      "network_type": "pstn",
      "time_limit": 600,
      "state": "connected",
```

```
      "date_created": "2026-01-19T08:02:32Z",
      "date_updated": "2026-01-19T08:03:06Z",
      "start_time": "2026-01-19T08:02:35Z",
      "additional_info": {
        "recording_status": "not_recording",
        "hold_status": "normal",
        "gather_status": "not_gathering",
        "play_status": "not_playing",
        "bridge_status": "not_bridged"
      }
    }
  }
}
```


# Response Fields

## Core Fields

| Field | Description |
|---|---|
| leg_sid | Unique identifier for the leg |
| account_sid | Exotel account identifier |
| contact_uri | Destination of the leg (phone/SIP URI) |
| exophone | Caller ID / Exotel number used |
| caller_id | Resolved caller ID presented |
| direction | inbound or outbound |
| network_type | pstn or voip |
| time_limit | Max time allowed for the leg |
| state | Current leg state (example: connected, terminal) |
| date_created | When leg was created |
| date_updated | Last update time |
| start_time | When leg connected/started (if applicable) |

## additional_info (only if fields requested)

| Field | Description |
|---|---|
| recording_status | Recording status |
| hold_status | Hold status |
| gather_status | DTMF gather status |
| play_status | Playback status |
| bridge_status | Bridge connection status |
| tx_audio_status | Transmit audio status (if requested) |
| rx_audio_status | Receive audio status (if requested) |

## Error Responses

| HTTP Code | Meaning | Description |
|---|---|---|
| 400 | Bad Request | Invalid fields value or malformed request |
| 401 | Unauthorized | Invalid API Key / Token |
| 403 | Forbidden | Access not allowed for this account |
| 404 | Not Found | Leg SID not found |
| 429 | Too Many Requests | Rate limit exceeded |
| 500 | Server Error | Internal error |

## Notes & Best Practices

- Use Get Leg Details for **polling** only when needed – prefer events for real-time status
- Always log request_id for debugging
- If you need runtime action statuses (play/record/bridge/hold), always pass fields

# 3.4.3. GET – Get Events for a Leg

## Description

Retrieve all external events generated for a specific **Leg**, including lifecycle events and action-level events.

This API is useful for debugging, auditing, and reconstructing the event timeline for a leg.

### Method

**GET**

## Endpoint

```
/v2/accounts/{AccountSID}/legs/{LegSID}/external_events
```

**Full URL**

```
https://cpaas-api.in.exotel.com/v2/accounts/{AccountSID}/legs/{LegSID}/external_events
```

## Authentication

Use **Basic Authentication**.

```
Authorization: Basic <BASE64_APIKEY_APITOKEN>
```

API credentials can be found in **my.exotel.com → Developer Settings**.

## Request Query Parameters

| Parameter Name | Mandatory / Optional | Type | Description |
|---|---|---|---|
| event_name | Optional | String | Filters events by a specific event name (e.g., leg_answered, recording_started) |

## Sample Request

```
GET /v2/accounts/{AccountSID}/legs/{LegSID}/external_events
```

### Sample Request with Filter

```
GET /v2/accounts/{AccountSID}/legs/{LegSID}/external_events?event_name=leg_answered
```

## Response

### Response Parameters

| Parameter Name | Type | Description |
|---|---|---|
| sid | string | Unique SID of the leg |
| event_type | string | Type of event (leg_action_event, leg_lifecycle_event) |
| account_sid | string | Account SID associated with the leg |
| external_events | array | List of external event objects |

### External Event Object

Each item in external_events contains:

| Field | Type | Description |
|---|---|---|
| endpoint_response | boolean | Whether the event was successfully delivered |
| error | string | Error message, if delivery failed |
| date_created | string (ISO-8601) | Event creation timestamp |
| date_updated | string (ISO-8601) | Event update timestamp |
| event_info | object | Detailed event metadata |

### Event Info Object

| Field | Type | Description |
|---|---|---|
| event_sid | string | Unique SID of the event |
| event_type | string | Event type (leg_action_event, leg_lifecycle_event) |
| event_name | string | Name of the event |
| event_timestamp | object | Timestamp when the event occurred |
| action_sid | string | Associated action SID |
| action_custom_parameter | string | Custom parameter passed during action execution |
| data | object | Event-specific payload |

## Sample Response


```
{
  "request_id": "ce7b872225b348f09e80115c1ff569a3",
  "method": "GET",
  "http_code": 202,
  "response": {
    "error_data": null,
    "data": {
      "sid": "2q47dakkScvff1IbPdpp7KJSe6j00000",
      "event_type": "leg_lifecycle_event",
      "account_sid": "sprinklr2m",
      "external_events": [
        {
          "endpoint_response": true,
          "error": "",
```

```
          "date_created": "2024-12-11T09:41:21.98089381Z",
          "date_updated": "2024-12-11T09:41:22.186124012Z",
          "event_info": {
            "event_sid": "2q47dipYPQbEnwgEnNWxrTB6YWb00000",
            "event_type": "leg_lifecycle_event",
            "event_name": "channel_created",
            "event_timestamp": {
              "seconds": 1733148727,
              "nanos": 963868626
            },
            "action_sid": "2pfEO9kg9YCqUJlbGHjYH2GaJa400000",
            "action_custom_parameter": "START_UNI_STREAM_ACTION_CUSTOM_PARAM",
            "data": {
              "account_sid": "sprinklr2m",
              "caller_id": "08030752501",
              "contact_uri": "09163816622",
              "custom_param": "abc",
              "date_created": "2024-12-11T09:41:21.98089381Z",
              "date_updated": "2024-12-11T09:41:22.186124012Z",
              "dc_code": "101",
              "direction": "outbound",
              "exophone": "08030752400",
              "leg_sid": "2q47dakkScvff1IbPdpp7KJSe6j00000",
              "network_type": "pstn",
              "nso_code": "a8140aaa-ce40-11ed-9a09-0242ac110003",
              "start_time": "2024-12-11T09:41:22.186124012Z",
              "state": "created",
              "time_limit": 14400
            }
          }
        }
      ]
    }
  }
}
```


## Notes for Developers

- Events are returned in **chronological order**
- This API is **read-only** and does not affect leg execution
- Useful for:
  - Debugging failed actions
  - Verifying event delivery
  - Building event-replay or analytics pipelines

# 3.4.4. Leg Actions

## Description

Leg Actions allow you to **control an existing leg** after it has been created. Using these actions, you can answer, hang up, play audio, record, gather DTMF, stream audio, bridge calls, and more.

All leg actions are executed via a **single Actions API** using an **EXOML payload**.

### Authentication

- **Type:** HTTP Basic Authentication
- **Username:** API Key
- **Password:** API Token

### Request Headers

```
Content-Type: application/json
Authorization: Basic <base64(API_KEY:API_TOKEN)>
```

### Request Body Structure

| Field | Type | Required | Description |
|---|---|---|---|
| exoml | string | Yes | XML describing the action |
| action_custom_param | string | No | Custom value echoed back in events |

### Base URL

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}
```

**Example**

```
https://cpaas-api.in.exotel.com/v2/accounts/ameyo5m
```

### Common Endpoint

All Leg Actions are executed using:

**POST**

```
/v2/accounts/{account_sid}/legs/{leg_sid}/actions
```

**Full URL**

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions
```

### Common Request Body

| Parameter | Type | Required | Description |
|---|---|---|---|
| action_custom_param | string | No | Custom value echoed back in external events |
| exoml | string | Yes | EXOML XML defining the action |

## Common Response

**HTTP 202 – Accepted**

```
{
  "request_id": "abc123",
  "http_code": 202,
  "response": {
    "data": {
      "leg_sid": "LEG_SID"
    }
  }
}
```

## Common Error Responses (applies to all actions)

| HTTP Code | Description |
|---|---|
| 400 | Invalid EXOML or parameters |
| 401 | Authentication failed |
| 403 | Not authorized |
| 404 | Leg not found |
| 429 | Rate limit exceeded |
| 500 | Internal server error |

## External Events (General)

- Events are delivered asynchronously to the configured leg_event_endpoint (gRPC).
- Each action emits one or more action-specific events (listed in child pages).

# 3.4.4.1. Categorized Overview

## Introduction

Leg Actions are executed on an active **call leg** to control **media**, **call state**, **routing**, **recording**, and **real-time integrations**.

Use this page to **quickly discover the right action** before diving into detailed API reference pages.

### A. Media & Interaction Actions

These actions control **what the participant hears, says, or inputs** during a call.

- **<StartPlay>** Plays an audio file from a URL to the leg.
- **<StopPlay>** Stops an ongoing audio playback.
- **<Say>** Converts text to speech and plays it to the leg. Supports **Amazon Polly** and **Google TTS**.
- **<StopSay>** Stops an ongoing text-to-speech playback.
- **<PlaySequence>** Plays a sequence of media actions (e.g., Play → Say → Play) in order.
- **<Gather>** Collects DTMF digits (for IVR menus, PIN entry, confirmations).
- **<SendDigits>** Plays DTMF tones *out* to the leg (used to interact with downstream IVRs).

### B. Call State Control

These actions manage the **lifecycle and audio state** of a call leg.

- **<Answer>** Picks up an incoming leg.
- **<Hangup>** Terminates the leg immediately.
- **<Mute> / <UnMute>** Mutes or unmutes audio on the leg. Supports direction control: in, out, or both.
- **<Hold> / <UnHold>** Places the leg on hold and resumes it. Supports optional **Music on Hold (MOH)** using nested <StartPlay>.

### C. Connectivity & Routing

Advanced actions for **connecting legs, conferencing, and supervision**.

- **<Dial>** Creates a new outbound leg and bridges it to the current leg. Supports PSTN and VoIP destinations.
- **<JoinBridge>** Moves the leg into an existing multi-party bridge.
- **<LeaveBridge>** Removes the leg from a bridge.
- **<StartMonitoring>** Allows a supervisor to **listen** or **whisper** on another leg (rxAudio: muted / enabled).
- **<StopMonitoring>** Stops an active monitoring session.

### D. Recording Actions

These actions manage **call recordings** at the leg level.

- **<StartRecording>** Starts recording audio on the leg. Supports multiple formats and storage options (S3 / HTTPS).
- **<StopRecording>** Stops an active recording and finalizes the recording file.

## E. Streaming & Real-Time Integrations

Used for **voice bots, real-time analytics, and AI processing**.

- **<StartStream>** Streams live audio from the leg to a WebSocket endpoint.Supports unidirectional and bidirectional streaming.
- **<StopStream>** Stops an active audio stream.

## F. Control & Event Routing

Actions that affect **control ownership and event delivery**.

- **<Refer>** Updates the external **event delivery endpoint (gRPC)** for the leg.Used when control needs to be transferred between systems.

## Quick Summary Table

| Category | Actions |
|---|---|
| Media & Interaction | StartPlay, StopPlay, Say, StopSay, PlaySequence, Gather, SendDigits |
| Call State Control | Answer, Hangup, Mute, UnMute, Hold, UnHold |
| Connectivity & Routing | Dial, JoinBridge, LeaveBridge, StartMonitoring, StopMonitoring |
| Recording | StartRecording, StopRecording |
| Streaming | StartStream, StopStream |
| Control & Event Routing | Refer |

# 3.4.4.2. Request, cURL & Response Payloads

## Endpoint (All Actions)

**POST**

```
/v2/accounts/{account_sid}/legs/{leg_sid}/actions
```

**Full URL**

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions
```

## Authentication

```
Authorization: Basic <BASE64_APIKEY_APITOKEN>
Content-Type: application/json
```

## <Answer>

### Request Payload

```
{
  "action_custom_param": "answer_call",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Answer></Answer></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "answer_call",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Answer></Answer></Flow>"
}'
```

## <Dial>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| contactUri | Yes | Destination number or SIP URI |
| exophone | Yes | Caller ID |
| networkType | No | pstn or voip |
| absorbDtmf | No | Absorb DTMF during dial |

### Request Payload

```
{
  "action_custom_param": "test_dial",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Dial contactUri=\"9791111865\"
exophone=\"02048566335\" networkType=\"pstn\" absorbDtmf=\"true\"></Dial></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "test_dial",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Dial contactUri=\"9791111865\"
exophone=\"02048566335\" networkType=\"pstn\" absorbDtmf=\"true\"></Dial></Flow>"
}'
```

## <Gather>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| numDigits | No | Number of digits to collect |
| timeoutInSec | No | Timeout duration |
| finishOnKey | No | Key to terminate input |

### Request Payload

```
{
  "action_custom_param": "collect_digits",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Gather numDigits=\"4\" timeoutInSec=\"10\"
finishOnKey=\"#\"></Gather></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "collect_digits",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Gather numDigits=\"4\" timeoutInSec=\"10\"
finishOnKey=\"#\"></Gather></Flow>"
}'
```

## <Hangup>

### Request Payload

```
{
  "action_custom_param": "hangup_call",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Hangup></Hangup></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "hangup_call",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Hangup></Hangup></Flow>"
}'
```

## <Hold>

### Request Payload

```
{
  "action_custom_param": "hold_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Hold></Hold></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "hold_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Hold></Hold></Flow>"
}'
```

## <JoinBridge>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| bridgeSid | Yes | Bridge identifier |
| absorbDtmf | No | Absorb DTMF |

### Request Payload

```
{
  "action_custom_param": "join_bridge",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><JoinBridge bridgeSid=\"BRIDGE_SID\"
absorbDtmf=\"true\"></JoinBridge></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "join_bridge",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><JoinBridge bridgeSid=\"BRIDGE_SID\"
absorbDtmf=\"true\"></JoinBridge></Flow>"
}'
```

## <LeaveBridge>

### Request Payload

```
{
  "action_custom_param": "leave_bridge",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><LeaveBridge></LeaveBridge></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "leave_bridge",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><LeaveBridge></LeaveBridge></Flow>"
}'
```

## <Mute> / <UnMute>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| direction | No | in, out, both |

### Request Payload

```
{
  "action_custom_param": "mute_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Mute direction=\"out\"></Mute></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "mute_leg",
```

```
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Mute direction=\"out\"></Mute></Flow>"
}'
```

# <PlaySequence>

## Request Parameters

Nested media actions only.

## Request Payload

```
{
  "action_custom_param": "play_sequence",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><PlaySequence>
<StartPlay>https://example.com/audio.wav</StartPlay><Say>Hello</Say></PlaySequence></Flow>"
}
```

## cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "play_sequence",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><PlaySequence>
<StartPlay>https://example.com/audio.wav</StartPlay><Say>Hello</Say></PlaySequence></Flow>"
}'
```

# <Refer>

## Request Parameters

| Attribute | Required | Description |
|---|---|---|
| legEventEndpoint | Yes | gRPC endpoint |

## Request Payload

```
{
  "action_custom_param": "refer_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Refer legEventEndpoint=\"grpc://host:port\"></Refer>
</Flow>"
}
```

## cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "refer_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Refer legEventEndpoint=\"grpc://host:port\"></Refer>
```

```
</Flow>"
}'
```

## <Say>

### Request Payload

```
{
  "action_custom_param": "say_text",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Say>Hello</Say></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "say_text",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Say>Hello</Say></Flow>"
}'
```

## <SendDigits>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| duration | No | Tone duration (ms) |
| between | No | Gap between digits |

### Request Payload

```
{
  "action_custom_param": "send_digits",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><SendDigits duration=\"100\" between=\"500\">1234#
</SendDigits></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "send_digits",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><SendDigits duration=\"100\" between=\"500\">1234#
</SendDigits></Flow>"
}'
```

## <StartMonitoring>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| monitoringLegSid | Yes | Target leg SID |
| rxAudio | No | muted / enabled |

### Request Payload

```
{
  "action_custom_param": "start_monitoring",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartMonitoring monitoringLegSid=\"LEG_SID\"
rxAudio=\"muted\"></StartMonitoring></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "start_monitoring",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartMonitoring monitoringLegSid=\"LEG_SID\"
rxAudio=\"muted\"></StartMonitoring></Flow>"
}'
```

## <StartPlay>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| Body (URL) | Yes | Public or authenticated audio file URL |

### Request Payload

```
{
  "action_custom_param": "start_play",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartPlay>https://example.com/audio.wav</StartPlay>
</Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "start_play",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartPlay>https://example.com/audio.wav</StartPlay>
</Flow>"
}'
```

# <StartRecording>

## Request Parameters

| Attribute | Required | Description |
|---|---|---|
| direction | No | in, out, both |
| format | No | wav, mp3 |
| storageType | No | https, s3 |
| storageURL | No | Upload endpoint |

## Request Payload

```
{
  "action_custom_param": "start_recording",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartRecording direction=\"both\" format=\"wav\"
storageType=\"https\" storageURL=\"https://example.com/upload\"></StartRecording></Flow>"
}
```

## cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "start_recording",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartRecording direction=\"both\" format=\"wav\"
storageType=\"https\" storageURL=\"https://example.com/upload\"></StartRecording></Flow>"
}'
```

# <StartStream>

## Request Parameters

| Attribute | Required | Description |
|---|---|---|
| streamType | Yes | unidirectional / bidirectional |
| streamUrl | Yes | WebSocket endpoint |

## Request Payload

```
{
  "action_custom_param": "start_stream",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartStream streamType=\"bidirectional\"
streamUrl=\"wss://example.com/stream\"></StartStream></Flow>"
}
```

## cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "start_stream",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartStream streamType=\"bidirectional\"
streamUrl=\"wss://example.com/stream\"></StartStream></Flow>"
}'
```

## <StopMonitoring>

### Request Payload

```
{
  "action_custom_param": "stop_monitoring",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopMonitoring></StopMonitoring></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "stop_monitoring",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopMonitoring></StopMonitoring></Flow>"
}'
```

## <StopPlay>

### Request Payload

```
{
  "action_custom_param": "stop_play",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopPlay></StopPlay></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "stop_play",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopPlay></StopPlay></Flow>"
}'
```

## <StopRecording>

### Request Payload

```
{
  "action_custom_param": "stop_recording",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopRecording></StopRecording></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "stop_recording",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopRecording></StopRecording></Flow>"
}'
```

## <StopSay>

### Request Payload

```
{
  "action_custom_param": "stop_say",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopSay></StopSay></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "stop_say",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopSay></StopSay></Flow>"
}'
```

## <StopStream>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| streamSid | Yes | Stream identifier |

### Request Payload

```
{
  "action_custom_param": "stop_stream",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopStream streamSid=\"STREAM_SID\"></StopStream>
</Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "stop_stream",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopStream streamSid=\"STREAM_SID\"></StopStream>
</Flow>"
}'
```

# <UnHold>

## Request Payload

```
{
  "action_custom_param": "unhold_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><UnHold></UnHold></Flow>"
}
```

## cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "unhold_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><UnHold></UnHold></Flow>"
}'
```

# <UnMute>

## Request Parameters

| Attribute | Required | Description |
|---|---|---|
| direction | No | in, out, both |

## Request Payload

```
{
  "action_custom_param": "unmute_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><UnMute direction=\"out\"></UnMute></Flow>"
}
```

## cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/legs/{leg_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "unmute_leg",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><UnMute direction=\"out\"></UnMute></Flow>"
}'
```

# 3.4.4.3. Leg Actions – External Events

## Introduction

This page lists all **external events emitted by Leg Actions.** Events are delivered asynchronously to the configured **leg event endpoint**.

### Answer

| Event Name | Event Type | Description |
|---|---|---|
| leg_answered | leg_action_event | Triggered when the leg is successfully answered |
| answer_failed | leg_action_event | Triggered when the answer action fails |

### Hangup

| Event Name | Event Type | Description |
|---|---|---|
| leg_terminated | leg_lifecycle_event | Triggered when the leg is terminated |
| hangup_failed | leg_action_event | Triggered when hangup fails |

### Dial

| Event Name | Event Type | Description |
|---|---|---|
| dial_initiated | leg_action_event | Triggered when dial action starts |
| dial_success | leg_action_event | Triggered when dial is successful |
| dial_completed | leg_action_event | Triggered when dial flow completes |
| dial_failed | leg_action_event | Triggered when dial fails |

### StartPlay

| Event Name | Event Type | Description |
|---|---|---|
| play_started | leg_action_event | Triggered when playback starts |
| play_completed | leg_action_event | Triggered when playback completes |
| play_interrupted | leg_action_event | Triggered when playback is interrupted |
| play_failed_to_start | leg_action_event | Triggered when playback fails to start |

### StopPlay

| Event Name | Event Type | Description |
|---|---|---|
| play_interrupted | leg_action_event | Triggered when playback is stopped |
| play_failed_to_stop | leg_action_event | Triggered when playback fails to stop |

## StartSay (Say)

| Event Name | Event Type | Description |
|---|---|---|
| say_started | leg_action_event | Triggered when TTS starts |
| say_completed | leg_action_event | Triggered when TTS completes |
| say_interrupted | leg_action_event | Triggered when TTS is interrupted |
| say_failed_to_start | leg_action_event | Triggered when TTS fails to start |

## StopSay

| Event Name | Event Type | Description |
|---|---|---|
| say_interrupted | leg_action_event | Triggered when TTS is stopped |
| say_failed_to_stop | leg_action_event | Triggered when stopping TTS fails |

## Gather

| Event Name | Event Type | Description |
|---|---|---|
| gather_initiated | leg_action_event | Triggered when gather is initiated |
| gather_started | leg_action_event | Triggered when digit collection starts |
| gather_success | leg_action_event | Triggered when digits are successfully collected |
| gather_failed | leg_action_event | Triggered when gather fails |
| gather_interrupted | leg_action_event | Triggered when gather is interrupted |

## SendDigits

| Event Name | Event Type | Description |
|---|---|---|
| digits_sent | leg_action_event | Triggered when DTMF digits are sent |
| digits_failed_to_send | leg_action_event | Triggered when DTMF sending fails |

## StartRecording

| Event Name | Event Type | Description |
|---|---|---|
| recording_started | leg_action_event | Triggered when recording starts |
| recording_stopped | leg_action_event | Triggered when recording stops |
| recording_failed_to_start | leg_action_event | Triggered when recording fails to start |
| recording_available | leg_action_event | Triggered when recording file is available |

## StopRecording

| Event Name | Event Type | Description |
|---|---|---|
| recording_stopped | leg_action_event | Triggered when recording stops |
| recording_failed_to_stop | leg_action_event | Triggered when stopping recording fails |
| recording_available | leg_action_event | Triggered when recording file is available |

## Hold

| Event Name | Event Type | Description |
|---|---|---|
| hold_success | leg_action_event | Triggered when leg is placed on hold |
| hold_failed | leg_action_event | Triggered when hold fails |

## UnHold

| Event Name | Event Type | Description |
|---|---|---|
| unhold_success | leg_action_event | Triggered when leg is resumed |
| unhold_failed | leg_action_event | Triggered when resume fails |

## Mute

| Event Name | Event Type | Description |
|---|---|---|
| mute_success | leg_action_event | Triggered when audio is muted |
| mute_failed | leg_action_event | Triggered when mute fails |

## UnMute

| Event Name | Event Type | Description |
|---|---|---|
| unmute_success | leg_action_event | Triggered when audio is unmuted |
| unmute_failed | leg_action_event | Triggered when unmute fails |

## JoinBridge

| Event Name | Event Type | Description |
|---|---|---|
| leg_joined_bridge | leg_action_event | Triggered when leg joins the bridge |
| leg_failed_to_join_bridge | leg_action_event | Triggered when joining bridge fails |

## LeaveBridge

| Event Name | Event Type | Description |
|---|---|---|
| leg_left_bridge | leg_action_event | Triggered when leg leaves the bridge |
| leg_failed_to_leave_bridge | leg_action_event | Triggered when leaving bridge fails |

## StartMonitoring

| Event Name | Event Type | Description |
|---|---|---|
| monitoring_initiated | leg_action_event | Triggered when monitoring is initiated |
| monitoring_started | leg_action_event | Triggered when monitoring starts |
| monitoring_failed | leg_action_event | Triggered when monitoring fails |

## StopMonitoring

| Event Name | Event Type | Description |
|---|---|---|
| monitoring_stopped | leg_action_event | Triggered when monitoring stops |
| monitoring_failed_to_stop | leg_action_event | Triggered when stopping monitoring fails |
| referred | leg_action_event | Triggered when control is transferred |

## StartStream

| Event Name | Event Type | Description |
|---|---|---|
| stream_initiated | leg_action_event | Triggered when streaming is initiated |
| stream_started | leg_action_event | Triggered when streaming starts |
| stream_stopped | leg_action_event | Triggered when streaming stops |
| stream_failed | leg_action_event | Triggered when streaming fails |

## StopStream

| Event Name | Event Type | Description |
|---|---|---|
| stream_stopped | leg_action_event | Triggered when streaming stops |
| stream_failed_to_stop | leg_action_event | Triggered when stopping stream fails |
| referred | leg_action_event | Triggered when control is transferred |

## Refer

| Event Name | Event Type | Description |
|---|---|---|
| referred | leg_action_event | Triggered when event endpoint is updated |

### Notes for Developers

- Events are **asynchronous**
- Receipt of a 202 Accepted response does **not** guarantee success
- Always rely on **external events** to determine action outcome

# 3.5. Rate Limits

## Introduction

To ensure platform stability and fair usage across all customers, Exotel enforces **rate limits** on API requests.

Rate limits apply **per account** and are evaluated based on **API type and request volume**.

## Default Limit

Default limit for all API is **100 RPM** which can be increased based on request to us. If you exceed the rate limit, you will receive a 429 HTTP response code. Please ensure that your application adheres to the rate limits to avoid disruptions in service.

## How Rate Limiting Works

- Rate limits are applied at the **account level**
- Limits may vary based on:
  - API category (Legs, Bridges, Actions, Events)
  - Region
  - Contractual plan
- Both **burst limits** and **sustained request limits** may apply

## Rate Limit Exceeded Response

If a request exceeds the allowed rate limit, the API responds with:

### HTTP Status Code

```
429 Too Many Requests
```

### Sample Response

```
{
  "request_id": "xyz123",
  "http_code": 429,
  "response": {
    "error": {
      "code": "RATE_LIMIT_EXCEEDED",
      "message": "Rate limit exceeded. Please retry after some time."
    }
  }
}
```

## Best Practices

- Implement **retry logic with exponential backoff**
- Avoid sending multiple concurrent actions on the same leg or bridge
- Batch workflows where possible instead of polling
- Use **external events (gRPC)** instead of frequent status checks

## Increasing Rate Limits

If your use case requires higher throughput (for example, large-scale outbound calling or real-time orchestration):

- Contact your **Exotel account manager**
- Share:
  - Expected request volume
  - API categories used
  - Peak traffic patterns

Custom rate limits may be provisioned based on use case and plan.

## Notes

- Rate limits are enforced to protect platform reliability
- Repeated violations may result in **temporary throttling**
- Rate limits apply uniformly across **production and sandbox environments**, unless explicitly stated otherwise

# 3.6. Bridge APIs - Introduction

A **Bridge** represents a multi-party audio conference that allows multiple **Legs** to be connected together.
Bridges are commonly used for **conferencing**, **agent handoffs**, **supervisor monitoring**, and **multi-leg call flows**.

Bridge APIs allow you to:

- Create a bridge
- Retrieve bridge details
- Update bridge configuration
- Dynamically join or remove legs using **Leg Actions**

## Key Concepts

- A bridge exists independently of legs
- A bridge becomes **active** when at least one leg joins
- Legs are added or removed using:
  - <JoinBridge>
  - <LeaveBridge>
- Bridge state changes are emitted as **external events**

## Base URL

```
https://cpaas-api.in.exotel.com/v2/accounts/
```

## Authentication

Use **Basic Authentication**.

```
Authorization: Basic <BASE64_APIKEY_APITOKEN>
Content-Type: application/json
```

API credentials can be found in **my.exotel.com → Developer Settings**.

## Related APIs

- **POST /bridges** – Create a bridge
- **GET /bridges/** – Get bridge details
- **PUT /bridges/** – Update bridge configuration
- **Leg Actions** – JoinBridge / LeaveBridge

## Notes for Developers

- Bridge updates apply only to **future participants**
- Active legs are not dropped when updating bridge configuration
- Use **JoinBridge / LeaveBridge** Leg Actions to manage participants

# Dialer Implementation Guide using Exotel Legs APIs

This guide shows you how to build an end-to-end predictive dialer on top of the Legs and Bridge APIs described above.

## Objective

This document provides a detailed, implementation-ready plan for building a scalable, real-time **predictive dialer** using Exotel's programmable **Legs and Bridge APIs**. It is designed to help you engage leads at scale, maximize agent productivity, reduce wait time, and ensure precision in call flow orchestration using Exotel as a programmable telephony layer.

## Approach Summary

- The customer's leg is dialed first using Exophones.
- Call events are streamed in real time via gRPC to detect pickup and leg status.
- Upon confirmation of pickup, an ideal agent is dynamically selected and dialled.
- The agent leg can also be pre-dialled and held live using our nailed-up connection.
- A waiting message (IVR) can be played to the customer until the agent is bridged.
- Once both legs are active, a bridge is created to establish a conversation.
- Calls can be recorded; all call metadata is accessible through the API and events.

**Retry logic, timeouts, and CRM disposition updates are managed by you.**

## Step-by-Step Implementation Using Exotel Legs APIs

## Step 1: Lead Selection and Dial Planning

Your backend dialer logic should dynamically determine whom to dial and when, using:

- Real-time agent availability
- CRM-driven lead prioritization
- Historical lead pickup trends and SLA commitments

**Responsibility:**
You are responsible for implementing this logic and using Exotel APIs to programmatically control all legs and session cycles.

## Step 2: Initiate Customer Leg

**API:** `POST /legs`

- Use an Exophone DID as the caller ID.
- Set `network_type` to `PSTN`.
- Attach a `leg_event_endpoint` (gRPC) for real-time event tracking.
- Set `timeout` and `time_limit` to define ringing and call duration.

**Expected Events:**
`leg_ringing`, `leg_answered`, `leg_terminated`

## Step 3: Handle Customer Leg Events

When the following is received:

- `leg_answered`

Then:

- Optionally play a message using `StartSay` or `StartPlay` (e.g., "Please hold...").
- Continue monitoring until agent leg is ready.

Ensure IVR playback is stopped once the bridge is established.

## Step 4A: Dial Agent Leg On-Demand

**API:** `POST /legs`

- Trigger agent leg only after the customer leg is answered.
- Use the same Exophone DID.
- Attach `leg_event_endpoint` to track agent leg events.
- **Important:** Use `reference_leg_sid` (customer leg SID) if your use-case involves connecting to the pre-defined agent. If you are allocating an agent dynamically on availability basis you can ignore this step.

JSON

```
{
  "contact_uri": "+91XXXXXXXXXX",
  "exophone": "080XXXXXXX",
  "reference_leg_sid": "2mF7xxxxxx",
  "custom_param": {
    "session_id": "sess-12345"
  }
}
```

**Expected Events:**
`leg_ringing`, `leg_answered`, `leg_terminated`

## Step 4B: (Optional) Pre-Dial Agent Leg

- Create the agent leg **before the customer leg**, with a high `time_limit` (e.g., 1800 seconds).
- Optionally play a hold tone or message using `StartSay`.
- Use `GET /legs/{agent_leg_sid}` to poll for leg status (`status = answered`).
- Once the customer answers, bridge them immediately with the pre-connected agent leg.

This reduces latency and is useful when predictive decisions are made in advance.

## Step 5: Create Bridge Between Legs

**API:** `POST /bridges`

- Use both `leg_sids` (agent and customer) to create a bridge.
- Attach `bridge_event_endpoint` to track `bridge_created`, `bridge_terminated`, and other events.

Stop any active IVR playback on either leg before the bridge is initiated.

### Step 6: Enable Call Recording (Optional)

**API:** `POST /bridges/{bridge_sid}/actions`

- Initiate recording at the bridge level.
- Exotel will emit `recording_available` events with URLs, metadata, and recording durations.

### Step 7: Post-Call Analytics and CRM Sync using events

**API:** `GET /legs/{leg_sid}`

- Retrieve:
  - Call duration
  - Termination reason
  - Recording links
- Push this data to your CRM for:
  - Lead status updates
  - Retry and follow-up logic
  - Agent performance scoring

### Best Practices and Implementation Considerations

| Area | Recommendation |
|---|---|
| Event Delivery | Use gRPC endpoints for low-latency, ordered event streams. |
| Bridging Consistency | Use `reference_leg_sid` to ensure connecting to a pre-defined agent with low latency. |
| IVR Playback Handling | Use `StartSay` or `StartPlay` after customer leg is answered; stop playback once the bridge is created. |
| Exophone Support | Fully supported for outbound PSTN legs; use as per your compliance and pickup-rate needs. |
| Leg Identifiers | Use `leg_sid` as the unique reference for all API interactions; Exotel does not use a unified CallSid model. |
| Retry Handling | Monitor `leg_failed_to_create`, `leg_terminated` |
| Session Tracking | Use `custom_param.session_id` consistently across legs and bridges for traceability. |
| Leg Status Polling | Use `GET /legs/{leg_sid}` to validate leg state (`initiated`, `ringing`, `answered`, `terminated`) when needed. |

### Conclusion

Using Exotel's Legs and Bridge APIs, you can deploy a powerful, scalable predictive dialer that supports:

- Fully API-driven orchestration of customer-agent calling
- Real-time control using gRPC event streams
- Flexible leg creation with IVR, AMD, and pre-connection support

- Clean session tracking and CRM sync using custom_param
- Future-readiness with nailed-up connections and intelligent retry APIs.

This design ensures operational efficiency, lead quality, and agent productivity—all while maintaining full control through Exotel's programmable voice stack. Please drop an email to **hello@exotel.com** or reach out to your account manager if you'd like support with sandbox setup, gRPC streaming samples, or reference flows to accelerate your implementation.

# 3.6.1. POST – Create a Bridge

## Description

Creates a new bridge that can be used to connect multiple legs.

### Method

**POST**

### Endpoint

```
/v2/accounts/{AccountSID}/bridges
```

### Request Parameters

| Parameter Name | Mandatory / Optional | Type | Description |
|---|---|---|---|
| bridge_type | Optional | string | Type of bridge (default: audio) |
| max_participants | Optional | number | Maximum number of legs allowed |
| recording | Optional | boolean | Enable or disable recording |

### Request Payload

```
{
  "bridge_type": "audio",
  "max_participants": 10,
  "recording": false
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{AccountSID}/bridges' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "bridge_type": "audio",
  "max_participants": 10,
  "recording": false
}'
```

### Response Payload

```
{
  "request_id": "abc123",
  "http_code": 201,
  "response": {
    "data": {
      "bridge_sid": "BRIDGE_SID",
      "bridge_type": "audio",
      "state": "created",
```

```
      "date_created": "2024-12-11T09:41:21Z"
    }
  }
}
```

# 3.6.2. GET – Get Bridge Details

## Description

Retrieves information about an existing bridge.

### Method

**GET**

### Endpoint

```
/v2/accounts/{AccountSID}/bridges/{BridgeSID}
```

### Request Parameters

*No request parameters*

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{AccountSID}/bridges/{BridgeSID}' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>'
```

### Response Payload

```
{
  "request_id": "def456",
  "http_code": 200,
  "response": {
    "data": {
      "bridge_sid": "BRIDGE_SID",
      "bridge_type": "audio",
      "state": "active",
      "participants": 2,
      "date_created": "2024-12-11T09:41:21Z"
    }
  }
}
```

# 3.6.3. PUT – Update a Bridge

## Description

Updates the configuration of an existing bridge.

### Method

**PUT**

### Endpoint

```
/v2/accounts/{AccountSID}/bridges/{BridgeSID}
```

### Request Parameters

| Parameter Name | Mandatory / Optional | Type | Description |
|---|---|---|---|
| recording | Optional | boolean | Enable or disable recording |
| max_participants | Optional | number | Update max participant limit |

### Request Payload

```
{
  "recording": true,
  "max_participants": 15
}
```

### cURL

```
curl --location --request PUT 'https://cpaas-api.in.exotel.com/v2/accounts/{AccountSID}/bridges/{BridgeSID}' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "recording": true,
  "max_participants": 15
}'
```

### Response Payload

```
{
  "request_id": "ghi789",
  "http_code": 200,
  "response": {
    "data": {
      "bridge_sid": "BRIDGE_SID",
      "recording": true,
      "max_participants": 15,
      "state": "active",
      "date_updated": "2024-12-11T10:02:11Z"
    }
```

```
  }
}
```

# 3.6.4. Bridge Actions

## Description

Bridge Actions allow you to control an existing **bridge** after it has been created.
Using these actions, you can play audio announcements, perform text-to-speech, start or stop recordings, and route bridge-level events to external systems.

Bridge Actions apply **to all legs currently joined to the bridge**, making them ideal for conferencing, announcements, and supervisory use cases.

All Bridge Actions are executed via a single **Actions API** using an **EXOML payload**.

## Authentication

**Type:** HTTP Basic Authentication

- **Username:** API Key
- **Password:** API Token

## Request Headers

```
Content-Type: application/json
Authorization: Basic <base64(API_KEY:API_TOKEN)>
```

## Request Body Structure

| Field | Type | Required | Description |
|---|---|---|---|
| exoml | string | Yes | XML describing the bridge action |
| action_custom_param | string | No | Custom value echoed back in external events |

## Base URL

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}
```

### Example

```
https://cpaas-api.in.exotel.com/v2/accounts/ameyo5m
```

## Common Endpoint

All Bridge Actions are executed using:

### POST

```
/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions
```

### Full URL

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions
```

## Common Request Body

| Parameter | Type | Required | Description |
|---|---|---|---|
| action_custom_param | string | No | Custom value echoed back in external events |
| exoml | string | Yes | EXOML XML defining the bridge action |

## Common Response

**HTTP 202 – Accepted**

```
{
  "request_id": "abc123",
  "http_code": 202,
  "response": {
    "data": {
      "bridge_sid": "BRIDGE_SID"
    }
  }
}
```

Bridge Actions are **asynchronous**.
Final success or failure is communicated via **external events**.

## Common Error Responses (applies to all bridge actions)

| HTTP Code | Description |
|---|---|
| 400 | Invalid EXOML or parameters |
| 401 | Authentication failed |
| 403 | Not authorized |
| 404 | Bridge not found |
| 429 | Rate limit exceeded |
| 500 | Internal server error |

## External Events (General)

- Events are delivered asynchronously to the configured **bridge_event_endpoint** (gRPC)
- Each bridge action emits one or more **action-specific events**
- Event details are documented in individual Bridge Action pages

## Supported Bridge Actions

- Refer
- StartPlay
- StopPlay
- StartSay
- StopSay
- StartRecording
- StopRecording

## Notes for Developers

- Bridge Actions affect **all participants** in the bridge
- Media actions apply to **current and future legs**
- Use **Leg Actions (JoinBridge / LeaveBridge)** to manage bridge membership
- Prefer Bridge Actions for **announcements and conferencing**, not individual control

# 3.6.4.1. Categorized Overview

## Description

Bridge Actions allow you to control **media playback, announcements, recordings, and event routing** for all participants connected to a bridge.
These actions apply **at the bridge level**, not to individual legs.

All Bridge Actions are executed using:

```
POST /v2/accounts/{AccountSID}/bridges/{BridgeSID}/actions
```

## A. Media & Interaction Actions

These actions control what **all participants in the bridge hear**.

### <StartPlay>

Plays an audio file (from a URL) to **every leg currently joined** to the bridge.

### <StopPlay>

Stops any currently playing audio on the bridge.

### <StartSay> (Say)

Converts text to speech and plays it to **all participants** in the bridge.

### <StopSay>

Stops an ongoing text-to-speech playback on the bridge.

## B. Recording Actions

These actions control **bridge-level recording**, capturing audio from all participants.

### <StartRecording>

Starts recording the bridge audio.
Supports configurable format and storage destinations.

### <StopRecording>

Stops the ongoing bridge recording and finalizes the recording file.

## C. Event Routing & Control

These actions control how **bridge events** are delivered to external systems.

### <Refer>

Registers a gRPC endpoint to receive **external events** for the bridge.
Used for real-time monitoring, analytics, or orchestration engines.

## Summary — Bridge Action APIs

| Category | Actions |
|---|---|
| Media & Interaction | StartPlay, StopPlay, StartSay, StopSay |
| Recording | StartRecording, StopRecording |
| Event Routing | Refer |

## Key Differences vs Leg Actions

| Aspect | Bridge Actions | Leg Actions |
|---|---|---|
| Scope | All participants | Single participant |
| Media playback | Broadcast | Individual |
| Recording | Conference-level | Per leg |
| Event routing | Bridge-wide | Per leg |

## Notes for Developers

- Bridge Actions are **asynchronous**
- Action success/failure is communicated via **Bridge External Events**
- Media actions affect **current and future participants**
- Use **Leg Actions (JoinBridge / LeaveBridge)** to manage membership

# 3.6.4.2. Request, cURL & Response Payloads

## Endpoint (All Bridge Actions)

**POST**

```
/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions
```

**Full URL**

```
https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions
```

## Authentication

```
Authorization: Basic <BASE64_APIKEY_APITOKEN>
Content-Type: application/json
```

## Common Response Payload (All Actions)

```
{
  "request_id": "abc123",
  "http_code": 202,
  "response": {
    "data": {
      "bridge_sid": "BRIDGE_SID"
    }
  }
}
```

```
Actions are asynchronous. Confirm success/failure via external events.
```

## <Refer>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| bridgeEventEndpoint | Yes | gRPC endpoint where bridge events should be delivered |

### Request Payload

```
{
  "action_custom_param": "refer_bridge",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Refer bridgeEventEndpoint=\"grpc://host:port\">
</Refer></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "refer_bridge",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Refer bridgeEventEndpoint=\"grpc://host:port\">
</Refer></Flow>"
}'
```

# <StartPlay>

## Request Parameters

| Attribute | Required | Description |
|---|---|---|
| Body (URL) | Yes | Public or authenticated audio file URL |

## Request Payload

```
{
  "action_custom_param": "start_play",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartPlay>https://example.com/audio.wav</StartPlay>
</Flow>"
}
```

## cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "start_play",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartPlay>https://example.com/audio.wav</StartPlay>
</Flow>"
}'
```

# <StopPlay>

## Request Payload

```
{
  "action_custom_param": "stop_play",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopPlay></StopPlay></Flow>"
}
```

## cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "stop_play",
```

```
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopPlay></StopPlay></Flow>"
}'
```

## <StartSay> (Say)

### Request Payload

```
{
  "action_custom_param": "start_say",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Say>Hello everyone</Say></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "start_say",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><Say>Hello everyone</Say></Flow>"
}'
```

## <StopSay>

### Request Payload

```
{
  "action_custom_param": "stop_say",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopSay></StopSay></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "stop_say",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopSay></StopSay></Flow>"
}'
```

## <StartRecording>

### Request Parameters

| Attribute | Required | Description |
|---|---|---|
| format | No | wav, mp3 |
| storageType | No | https, s3 |
| storageURL | No | Upload endpoint |

### Request Payload

```
{
  "action_custom_param": "start_recording",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartRecording format=\"wav\" storageType=\"https\"
storageURL=\"https://example.com/upload\"></StartRecording></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "start_recording",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StartRecording format=\"wav\" storageType=\"https\"
storageURL=\"https://example.com/upload\"></StartRecording></Flow>"
}'
```

## <StopRecording>

### Request Payload

```
{
  "action_custom_param": "stop_recording",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopRecording></StopRecording></Flow>"
}
```

### cURL

```
curl --location 'https://cpaas-api.in.exotel.com/v2/accounts/{account_sid}/bridges/{bridge_sid}/actions' \
--header 'Authorization: Basic <BASE64_APIKEY_APITOKEN>' \
--header 'Content-Type: application/json' \
--data '{
  "action_custom_param": "stop_recording",
  "exoml": "<?xml version=\"1.0\" encoding=\"UTF-8\"?><Flow><StopRecording></StopRecording></Flow>"
}'
```

# 3.6.4.3. Bridge Actions - External Events

## Description

This page lists **external events emitted for Bridge Actions**. Use these events to confirm whether a bridge action succeeded or failed.

### Refer (On Bridge)

| Event Name | Event Type | Description |
|---|---|---|
| referred | bridge_action_event | The URL was updated successfully. |

### StartPlay

| Event Name | Event Type | Description |
|---|---|---|
| play_started | bridge_action_event | The play started on the bridge. |
| play_completed | bridge_action_event | File has completely played out on the bridge and now play has stopped. |
| play_interrupted | bridge_action_event | Play was stopped manually by another action. |
| play_failed_to_start | bridge_action_event | The file failed to play on the bridge. |

### StopPlay

| Event Name | Event Type | Description |
|---|---|---|
| play_interrupted | bridge_action_event | Play was stopped successfully, before the file was completed. |
| play_failed_to_stop | bridge_action_event | StopPlay action was not completed. |

### StartSay

| Event Name | Event Type | Description |
|---|---|---|
| say_started | bridge_action_event | The say started on the bridge. |
| say_completed | bridge_action_event | Text has completely played out on the bridge and now say has stopped. |
| say_interrupted | bridge_action_event | Say was stopped manually by another action. |
| say_failed_to_start | bridge_action_event | The message failed to start playing out on the bridge. |

### StopSay

| Event Name | Event Type | Description |
|---|---|---|
| say_interrupted | bridge_action_event | Say was stopped successfully, before the message was played out. |
| say_failed_to_stop | bridge_action_event | StopSay action was not completed. |

### StartRecording

| Event Name | Event Type | Description |
| --- | --- | --- |
| recording_started | bridge_action_event | Indicates that the recording has started. |
| recording_stopped | bridge_action_event | Indicates that the recording has stopped. |
| recording_failed_to_start | bridge_action_event | Indicates a failure to start the recording process. |
| recording_available | bridge_action_event | Indicates that the recording is available. |

## StopRecording

| Event Name | Event Type | Description |
| --- | --- | --- |
| recording_failed_to_stop | bridge_action_event | Indicates a failure to stop the recording process. |
| recording_stopped | bridge_action_event | Indicates that the recording has stopped. |
| recording_available | bridge_action_event | Indicates that the recording is available. |

# 4. Voice SDKs

## 4.1. Which Voice SDK should I use?

### Which Voice SDK should I use?

Use this guide to pick the right Exotel Voice SDK for your product and how you manage users.

### Quick decision

| If your users call from... | Choose |
|---|---|
| A CRM or web portal, and you want Exotel to manage SIP users | **CRM WebSDK** |
| A custom browser softphone with your own UI | **Client WebSDK** |
| A native Android app | **Client Android SDK** |
| A native iOS app | **Client iOS SDK** |
| A Flutter (Android + iOS) app | **Client Flutter SDK** |

### SDK comparison

| SDK | Best for | Key capabilities | User management | Integration effort |
|---|---|---|---|---|
| **Client WebSDK** | Custom web softphone / agent desktop | Register, answer, hangup, mute, hold, DTMF, diagnostics | You manage users via **Subscriber Management API** and your backend | Medium – you own UI + credentials/token flow |
| **Client Android SDK** | Native Android apps | Outbound dial, incoming via push, mute/hold, call states | Subscriber Management API + `subscriber_token` from your backend | Medium–High – foreground service, FCM, permissions |
| **Client iOS SDK** | Native iOS apps | Same as Android for iPhone/iPad | Subscriber Management API + `subscriber_token` from your backend | Medium–High – PushKit/APNs, mic & notification permissions |
| **Client Flutter SDK** | One mobile codebase for Android + iOS | Cross-platform VoIP calling | Same subscriber + token model as native mobile SDKs | Medium – Flutter app + platform push setup |
| **CRM WebSDK** | Embedding calling inside CRM / web portals | Softphone widget, inbound & outbound, click-to-call, IP ↔ PSTN toggle, CRM SSO | Exotel manages SIP ID & password. Provision via **Customer** → **Application** → **Users** APIs. Use email as app user ID. | Lowest – pass auth token + user ID |

## User management comparison

| Topic | Client Web / Android / iOS / Flutter | CRM WebSDK |
|---|---|---|
| User model | Subscribers | Customer → Application → Users |
| Who creates SIP credentials? | You create subscribers; auth via subscriber token | Exotel (on user creation) |
| How agents start calling | Log in to your app → backend issues token → SDK initializes | Log in to CRM → SDK initializes & registers |
| Multiple teams (e.g. Sales vs Support) | Separate subscriber groups / backends as needed | Create separate **Applications** under the same Customer |
| Add users from Exotel dashboard? | Use Subscriber Management APIs | No – use APIs only |

## Recommended by use case

| Use case | Recommended SDK |
|---|---|
| Fully custom web dialer UI | **Client WebSDK** |
| Field agents on Android | **Client Android SDK** |
| Field agents on iOS | **Client iOS SDK** |
| Single mobile app for Android and iOS | **Client Flutter SDK** |
| Softphone inside HubSpot, Freshdesk, Zoho, Salesforce, or custom CRM | **CRM WebSDK** |
| Click-to-call from CRM contact pages | **CRM WebSDK** |
| Agents switch between IP and PSTN when network is poor | **CRM WebSDK** |

## Related docs

- **Voice SDKs**
- **IP-PSTN intermix: Customer onboarding and WebRTC SDK integration**
- **Subscriber Management API**

# 4.2. Client webSDK

## 4.2.1. Introduction

**Introduction**

Exotel Webrtc SDK library enables you to add the voip calling feature into your web app. This document outlines the integration and usage. The Exotel Webrtc SDK package is layered. We have web-client-sdk that provides higher level APIs and callbacks to register/unregister, receive calls and operate on calls like mute/unmute, hold/unhold. And we have webrtc-core-sdk that provides the underlying state machine and SIP protocol stack.

The current document provides API and Callback details for the web-client-sdk.

**Licensing**

You need an exotel account to use the voip calling functionality with this websdk. Contact exotel support for demo and account creation.

```
Exotel organization npm account : @exotel-npm-dev
```

For using the above organizational account for publishing , an invitation will be required. And for the same please contact the account manager or hello@exotel.com

**Glossary**

| Terminology | Description |
|---|---|
| App | Web Application |
| VOIP | Voice over IP |
| Client | User / Subscriber / Client signing up to use the web app |
| Customer | Exotel's customer licensing the SDK |

# 4.2.2. Getting Started

Software Package

The Exotel Webrtc SDK software package includes

- `@exotel-npm-dev/webrtc-client-sdk` library from npm repository
- Integration Guide
- Sample application for reference

Add Web Client SDK Library to Project

Install the compatible node.js and npm versions. For example, on ubuntu 20.04, following versions are compatible

- `nodejs version: >=14.x`
- `npm version: 6.14.4`

Once nodejs is installed, download the packages from the repository and install them in your npm modules directory. To download the library from the repository follow the steps given below,

Step 1: After successful login, install the webrtc-sdk library.

```
npm install @exotel-npm-dev/webrtc-client-sdk
```

Step 2: Verify if node_modules are created inside the sample app folder. Start the npm

Step 3: In order to launch the application, run the following command. Application should start on https://localhost:3000

```
npm start
```

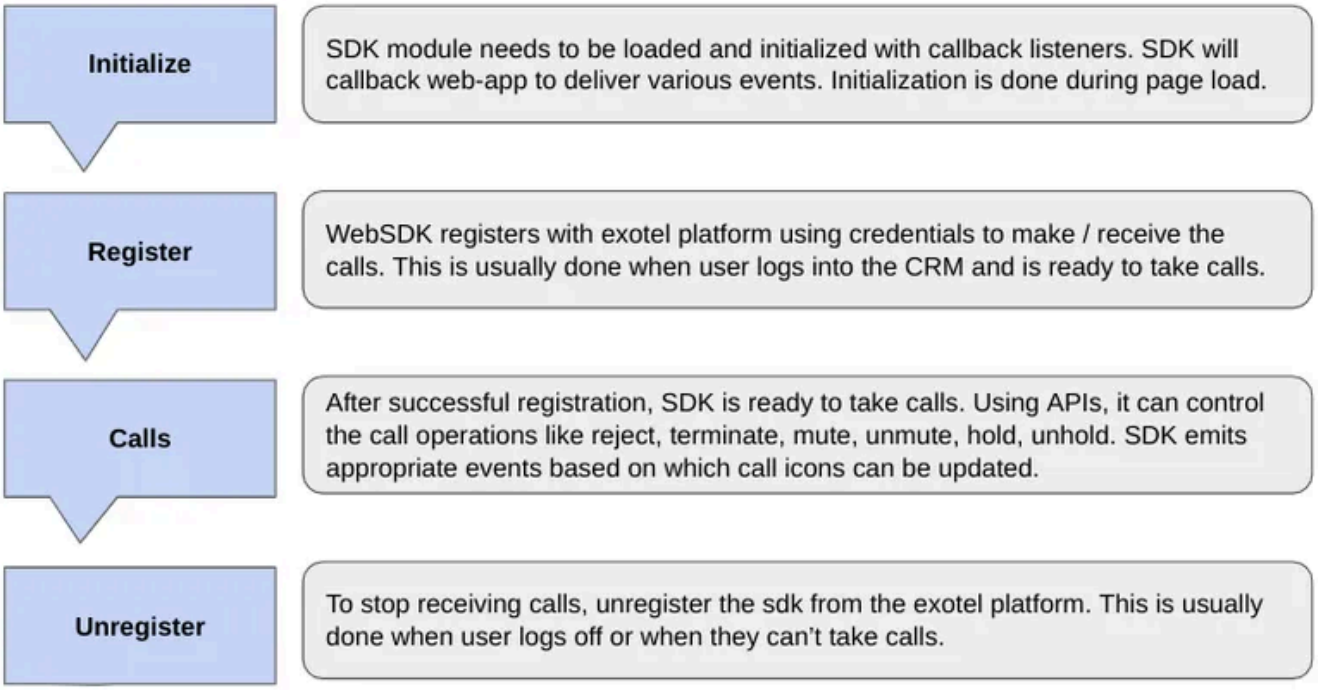
**Supported Browsers**

The SDK has been verified with the following browsers.

- Windows OS: Google Chrome, Mozilla Firefox and Microsoft Edge
- Linux OS: Google Chrome and Mozilla Firefox
- Mac OS: Google Chrome, Mozilla Firefox, Safari (>V11)

# 4.2.2.1. Web Client SDK APIs and Integration Workflow

### Web Client SDK APIs and Integration Workflow




### Initialize Library

WebRTC client SDK library needs to be initialized prior to using. A sample code snippet to do so is as below.

Step1: Import the exotel client library as below.

JS

```
import { ExotelWebClient as exWebClient } from '@exotel/webrtc-client-sdk';
```

Step2: Initialize the Exotel Web Client with SipAccountInfo and Callbacks using the API initWebrtc.

JS

```
// Initialise sipAccountInfo  dictionary
    const sipAccountInfo = {
      'userName': 'Username',
      'authUser': 'Username',
      'sipdomain': 'Domain',
      'domain': 'HostServer' + ":" + Port
      'displayname': 'DisplayName',
      'secret': 'Password',
      'port': 'Port',
      'security': 'wss',
    }

// Initialize Callbacks:

exWebClient.initWebrtc(sipAccountInfo,
                  RegisterEventCallBack,
                  CallListenerCallback,
                  SessionCallback)
```

This API also takes as input three callbacks:

- RegisterEventCallBack handles registration states as they change.
- CallListenerCallback handles call events as they occur.
- SessionCallback handles notifications for multiple tab sessions.

The details of the callbacks are explained in the corresponding flows.

Argument Details

| Args | DataType |
|---|---|
| sipAccountInfo | object |
| RegisterEventCallBack | Callback function |
| CallListenerCallback | Callback function |
| SessionCallback | Callback function |

| sipAccountInfo | | |
|---|---|---|
| authUser | String | SIP username to register. |
| userName | String | Used as a unique map index for phones. Same as authUser. |
| displayName | String | Local Displayname on Dialer. |
| secret | String | SIP Password |
| sipdomain | String | SIP  public domain |
| security | String | “wss”/”ws” typically “wss” |
| port | String | 443 for Websockets. |

## Opus Codec Preference - Optional

1. To enable opus codec, it can be enabled from Exotel Voip domain settings, for that we can raise a request to hello@exotel.com to enable the opus codec support.
2. Once opus codec is enabled, then and if browser is not preferring opus codec in SIP 200 OK, then we have an API to make opus codec as preferred codec.

JS

```
exWebClient.setPreferredCodec(“opus”);
```

## Download Logs - Optional

1. To help with debugging or sharing logs with support, whenever a user faces an issue, we can invoke the downloadLogs method.
2. This will download a “.txt” file (e.g. ”webrtc_sdk_logs_2025-04-03.txt” ) containing 1000 logs stored in the browser’s localStorage.

JS

```
exWebClient.downloadLogs();
```

## Register the SIP Phones

Before receiving / making any call, the SIP phone needs to be registered; and the API for this purpose is “DoRegister”.

| API Name | ExotelWebClient.DoRegister |
|---|---|
| Args | None |
| | |
| Exported To | Application UI |

JS

```
//Ensure that initWebRTC is invoked before calling DoRegister
exWebClient.DoRegister();
```

Once the registration has been done, the response comes in the registrationCallback with the events, “registered ”/ ”terminated” as below.

| API Name | RegisterEventCallBack | | |
|---|---|---|---|
| Args | **Params** | **Type** | **Values** |
| | state | String | "registered" / “terminated” / “sent_request” / “unregistered” |
| | phone | String | Username |
| Exported To | Application UI | | |

JS

```
function RegisterEventCallBack (state, phone){
    if (state === 'registered') {
// Successful registration
      setRegState(true)
    } else if (state === 'unregistered') {
// Successful unregistration
      setRegState(false)
    } else if (state === 'terminated')  {
// Registration/Unregistration failed
      setRegState(false)
    } else if (state === 'sent_request')  {
// Registration/Unregistration Request Sent
      if (unregisterWait === "true") {
        unregisterWait = "false";
        setRegState(false)
      }
    }
  }
```

## UnRegister the SIP Phones

To stop getting calls anymore, the SIP phones need to be unregistered. The API to do so is “UnRegister”.

| API Name | ExotelWebClient.UnRegister |
|---|---|
| Args | None |
| | |
| Exported To | Application UI |

JS

```
//Ensure that initWebRTC is invoked before calling DoRegister.
exWebClient.UnRegister();
```

The response to unregistration comes in the registrationCallback as below.

| API Name | RegisterEventCallBack | | |
|---|---|---|---|
| Args | **Params** | **Type** | **Values** |
| | state | String | "registered" / "unregistered" / “terminated” / “sent_request” |
| | phone | String | Username |
| Exported To | Application UI | | |

JS

```
  function RegisterEventCallBack (state, phone){
    if (state === 'registered') {
// Successful registration
      setRegState(true)
    } else if (state === 'unregistered') {
// Successful unregistration
      setRegState(false)
    } else if (state === 'terminated')  {
// Registration/Unregistration failed
      setRegState(false)
     } else if (state === 'sent_request')  {
// Registration/Unregistration Request Sent
      if (unregisterWait === "true") {
        unregisterWait = "false";
        setRegState(false)
      }
    }
  }
```

Note 1: If you have more than one phone, the second argument “phone” would give the indication as to which phone got unregistered.

....continue to **Web Client SDK APIs**

# 4.2.2.2. Web Client SDK APIs

## Receive Calls

Once the registration has happened, and when there is an incoming call, the callback registered for “Call Events” would get an event along with the details of the incoming call.

| API Name | CallListenerCallback | | |
|---|---|---|---|
| Args | **Params** | **Type** | **Values** |
| | callObj | Object | { callId, callState, callDirection, callStartedTime, remoteDisplayName }<br><br>callId {String}: sip call-id<br>callState {String}: incoming<br>callDirection {String}: incoming<br>callStartedTime {String}: call start time<br>remoteDisplayName {String}: callee name |
| | eventType | String | incoming : shows incoming call message<br>connected : open dialer<br>callEnded : close dialer<br>activeSession: session continuity |
| | phone | String | Username identifying the phone to which the call is coming. |
| Exported To | Application UI | | |

In the following snippet,the UI states are appropriately modified based on the call events.

JS

```
 function CallListenerCallback(callObj, eventType, phone) {
    if (eventType === 'incoming') {
// Incoming Call
      setCallComing(true)
    }  else if (eventType === 'connected') {
// Call in connected state
      setCallComing(false)
      setCallState(true)
    }  else if (eventType === 'callEnded') {
    // Call ended
      setCallComing(false)
      setCallState(false)
    } else if (eventType === 'terminated') {
// Call terminated
      setCallComing(false)
      setCallState(false)
    }
 }
```

## Accept Calls

Once the incoming call event is received, based on the user action the call can be accepted. The API to do so is “Call.Answer” as below. The call object could be obtained by invoking the “getCall()” method in exWebClient object.

| | |
|---|---|
| API Name | Call.Answer |
| Args | None |
| Exported To | Application UI |

JS

```
function acceptCallHandler() {
  call =  exWebClient.getCall()
  call.Answer();
}
```

The response event to accept calls comes in “CallListenerCallback”. Successful acceptance results in a “connected” event. Other possible events are:

“callEnded” : Call ended locally.

“terminated” : Call terminated remotely.

JS

```
function CallListenerCallback(callObj, eventType, phone) {
   if (eventType === 'incoming') {
// Incoming Call
      setCallComing(true)
   }  else if (eventType === 'connected') {
// Call in connected state
      setCallComing(false)
      setCallState(true)
   }  else if (eventType === 'callEnded') {
// Call ended
      setCallComing(false)
      setCallState(false)
   } else if (eventType === 'terminated') {
// Call terminated
      setCallComing(false)
      setCallState(false)
   }
}
```

## Hangup / Reject Calls

Once the incoming call event is received, based on the user action the call can be rejected. The API to do so is “Call.Hangup” as below.

| | |
|---|---|
| API Name | Call.Hangup |
| Args | None |
| Exported To | Application UI |

JS

```
function rejectCallHandler() {
  call =  exWebClient.getCall()
  call.Hangup();
}
```

For call hangup by local user, the response comes as a “callEnded” event. For call hangup by remote user,, the event comes as “terminated”.

JS

```
function CallListenerCallback(callObj, eventType, phone) {
    if (eventType === 'callEnded') {
// Call ended
      setCallComing(false)
      setCallState(false)
    } else if (eventType === 'terminated') {
// Call terminated
      setCallComing(false)
      setCallState(false)
    }
 }
```

## Mute / Unmute Calls

Once the call is in progress, based on the user action the call can be muted/unmuted. The APIs to do so are “Call.Mute” and “Call.UnMute” as below.

| | |
|---|---|
| API Name | Call.Mute |
| Args | None |
| Exported To | Application UI |

| | |
|---|---|
| API Name | Call.UnMute |
| Args | None |
| Exported To | Application UI |

You can also use a single API to toggle the mic using Call.MuteToggle.

| | |
|---|---|
| API Name | Call.MuteToggle |
| Args | None |
| Exported To | Application UI |

JS

```
function muteHandler() {
   call =  exWebClient.getCall()

   //call.MuteToggle(); // or
   if (!callOnMute) {
     call.Mute()
     callOnMute = true
   } else {
     call.UnMute()
     callOnMute = false
   }
 }
```

There is no callback event for mute. Call remains “connected” but with the mic muted. This is a toggle operation.

## Hold / Resume Calls

Once the call is in progress, based on the user action the remote user can be set on hold/unhold. The API to do so is “Call.Hold” and “Call.UnHold” as below.

| | |
|---|---|
| API Name | Call.Hold |
| Args | None |
| Exported To | Application UI |

| | |
|---|---|
| API Name | Call.UnHold |
| Args | None |
| Exported To | Application UI |

You can also use a single API to hold the remote caller using Call.HoldToggle.

JS

```
function holdHandler() {
    call =  exWebClient.getCall()

    //call.HoldToggle(); // or
    if (!callOnHold) {
      call.Hold()
      callOnHold = true
    } else {
      call.UnHold()
      callOnHold = false
    }
  }
```

There is no callback event for call hold. Call remains in “connected” but in sendonly mode. This is a toggle operation.

## Send DTMF

Once the call is in progress, based on the user action the remote user can be sent DTMF digits. The API to do so is “Call.sendDTMF” as below

| API Name | Call.sendDTMF |
|---|---|
| Args | **Params**<br>**Values**<br>**Type** |
| digit | String<br>0-9, A-D, *, # |
| Exported To | Application UI |

JS

```
call =  exWebClient.getCall()
if (call) {
call.sendDTMF(digit);
}
```

## Multitab Scenarios

When the webrtc sdk is loaded in multiple tabs then all the instances will register with the backend and receive incoming call alerts. There are two ways to avoid this,

- Maintain a single login session for the user in your webapp so that only one instance of the webrtc sdk is loaded. This is preferred.
- Maintain a parent-child relationship across the tabs in your webapp so that call is handled only in the parent tab.

Exotel Webrtc-SDK supports multiple tab sessions using Broadcast Channel. In this feature,  session listener and session callbacks can be used to get the indication on child tabs. A SessionListener creates a broadcast channel and sends a broadcast event to each

child tab for the following events, incoming / connected / callEnded / re-register. When a child tab (as per the logic maintained by the “Application Backend”.) receives a session event, it may notify but not handle the callback.

When the parent tab is destroyed, a re-register event comes to child tabs, based on the logic of the client, a child can opt to be the new parent.

Note 1: The logic of maintaining the parent and child tabs has to be in “Application Backend” by the “Customer”.

Note 2: If multitab scenario is not used, there is no need to handle the events in the session callback.

| API Name | SessionListener |
| --- | --- |
| Args | None |
| Exported To | Application UI |

```
SessionListener();  // To be called during initialization.
```

| API Name | SessionCallback | | |
|---|---|---|---|
| Args | **Params** | **Type** | **Values** |
| | callState | String | incoming / connected / callEnded / re-register<br><br>incoming :<br>child tab to show a notification message<br><br>connected :<br>child tab to close the notification message<br><br>callEnded :<br>child tab to close the notification message<br><br>re-register :<br>child tab to register when parent tab is closed<br><br>ice_gathering_state_<state>:<br>ICE gathering state changed. <state> will be the new ICE gathering state (e.g., new, gathering, complete).<br><br>ice_connection_state_<state>:<br>ICE connection state changed. <state> will be the new ICE connection state (e.g., new, checking, connected, disconnected, failed, closed).<br>Media_permission_denied: User denied media (microphone/camera) permissions. |
| | phone | String | username identifying the phone to which the call is coming. |
| Exported To | Application UI | | |

A sample code snippet for session callback is as below.

JS

```
function SessionCallback(callState, phone) {
      /**
       * SessionCallback is triggered whenever an incoming call arrives
       * which needs to be handled across tabs
       */
      switch(callState){
        case 'incoming':
console.log('incoming call' + phone)
            /**
             * Display a different notification popup in case of child tabs
             */
if(window.sessionStorage.getItem('activeSessionTab') !== 'parent0'){
            const message = 'Incoming call from ' + phone + ' ,Switch tab to find dialpad'
            setMessage(message);
          }
        break;
        case 'callEnded':
          /**
           * When call is either accepted or rejected then this is gets shutdown
           */
          console.log('call ended' + phone)
          setOpen(false);
        break;
        case 'connected':
          /**
           * When call is connected close the notification popup on child tabs
           */
          console.log('call connected' + phone)
          setOpen(false);
        break;
        case 're-register':
          /**
           * In case if the main/parent tab is closed then make the subsequent tab in the tab list as the parent tab
           * and send register for the same and also make that tab as the master
           */
    // ICE Gathering State Events
        case 'ice_gathering_state_new':
        case 'ice_gathering_state_gathering':
        case 'ice_gathering_state_complete':
            console.log('ICE state change:', callState, 'for call from:', phone);
            // Handle ICE state changes as needed
            break;

        // Media Permission Error
        case 'media_permission_denied':
            console.log('Media permission denied for call from:', phone);
            showErrorMessage('Microphone access is required for calls. Please allow microphone permissions.');
            break;
            window.sessionStorage.removeItem('activeSessionTab');
            window.sessionStorage.setItem('activeSessionTab', 'parent0');
            sendAutoRegistration();
          }
          break;
      }
  };
```

## Device and Network Diagnostics

**Initialize diagnostics.**

Diagnostics can be initialized by passing two callbacks, one troubleshooting logs and one to get the responses of diagnostics back.

| API Name | ExotelWebClient.initDiagnostics | | |
|---|---|---|---|
| Args | **Params** | **Type** | **Values** |
| | diagnosticsReportCallback | Object | Defined below |
| | keyValueSetCallback | Object | Defined below |
| Exported To | Application UI | | |

The signature of the callbacks are as below.

diagnosticsReportCallback is for logs

JS

```
function diagnosticsReportCallback(logStatus, logData) {
 // logStatus : Additional information on the logs, can be ignored as of now.
 // logData : Troubleshooting log to save in a file..
}
```

diagnosticsKeyValueCallback is for test responses, described in the following sections.

JS

```
function diagnosticsKeyValueCallback(key, status, description) {
  // key : Indicates the type of response
  // status: the value/status specific to key
  // description : description specific to key
}
```

Immediately after invoking the initDiagnostics, three parameters are returned through the diagnosticsKeyValueCallback callback.

| | |
|---|---|
| browserVersion | browserName/browserVersion.<br>Eg. Chrome/101.0.0.0 |
| micInfo | Mic name returned by the browser.<br>Eg. “Built-in Audio Analog Stereo” |
| speakerInfo | Speaker Name returned by the browser.<br>Eg. “Built-in Audio Analog Stereo” |

**Speaker Test**

Speaker test can be started by invoking the API “startSpeakerDiagnosticsTest”.

| | |
|---|---|
| API Name | ExotelWebClient.startSpeakerDiagnosticsTest |
| Args | None |
| Exported To | Application UI |

This starts a test by playing a ringtone. And as the ring tone gets played, the volume levels are passed through the callback function “diagnosticsKeyValueCallback” which can then be used to render the UI to show volume meter.

JS

```
function diagnosticsKeyValueCallback(key, status, description) {
  //key:“speaker”
  //status: a floating point value
  //description : "speaker ok"/”speaker error”
}
```

Once the user response is captured, it can be passed back to the API, “stopSpeakerDiagnosticsTest " as below. The responses are passed as arguments. THe

| | |
|---|---|
| API Name | ExotelWebClient.stopSpeakerDiagnosticsTest |
| Args | Optional, if present, “yes”/”no”. |
| Exported To | Application UI |

The response “yes” indicates that the user has heard the speaker's sound. “No” indicates that the user did not hear the speaker's sound. This response is further used to update the troubleshooting logs.

If no arguments are passed, only the test is terminated. No updates would be made to troubleshooting logs.

**Mic Test**

Mic test can be started by invoking  the API “startMicDiagnosticsTest”.

| | |
|---|---|
| API Name | ExotelWebClient.startMicDiagnosticsTest |
| Args | None |
| Exported To | Application UI |

This starts a test by capturing the audio spoken on the mic. And as the audio is analyzed, the volume levels are passed through the callback function “diagnosticsKeyValueCallback”  with key as “mic” which can then be used to render the UI to show mic volume meter.

JS

```
function diagnosticsKeyValueCallback(key, status, description) {
    //key:“mic”
    //status: a floating point value
    //description : "mic ok"/”mic error”
  }
```

Once the user response is captured, it can be passed back to the API, “stopMicDiagnosticsTest " as below. The responses are passed as arguments. THe

| | |
|---|---|
| API Name | ExotelWebClient.stopMicDiagnosticsTest |
| Args | Optional, if present, “yes”/”no”. |
| Exported To | Application UI |

The response “yes” indicates that the user has been captured by the mic. “No” indicates that the user's voice could not be captured. This response is further used to update the troubleshooting logs.

If no arguments are passed, only the test is terminated. No updates would be made to troubleshooting logs.

**Network Diagnostics**

Network diagnosis can be started by invoking  the API “startNetworkDiagnostics”.

| | |
|---|---|
| API Name | ExotelWebClient.startNetworkDiagnostics |
| Args | None |
| Exported To | Application UI |

This API starts network operations testing. The callback “diagnosticsKeyValueCallback” is called with appropriate keys after each test completion.

## Web Socket Connection Callback

Returns a WSS url with status “connected” on successful connectivity.

JS

```
function diagnosticsKeyValueCallback(key, status, description) {
  // key:“wss”
  // status = connected/disconnected
  // description = WSS URL
}
```

## User Registration Status Callback

Returns a status “connected” on successful registration of the configured “username”. This is the callback from the background registration requests. No explicit registration requests are sent specifically for diagnostics purposes.

JS

```
function diagnosticsKeyValueCallback(key, status, description) {
  // key:“userReg”
  // status - "registered"/"unregistered"
  // description - userName
}
```

## TCP connectivity callback

Returns a key value “tcp”with ice candidate information as description.

JS

```
function diagnosticsKeyValueCallback(key, status, description) {
   // key:“tcp”
   // status: connected/disconnected
   // description : ice candidate line for tcp connectivity/empty string
  }
```

## UDP connectivity callback

Returns a key value “udp” with ice candidate information as description.

JS

```
function diagnosticsKeyValueCallback(key, status, description) {
  // key:“udp”
  // key: connected/disconnected
  // description : ice candidate line for udp connectivity/empty string
}
```

## Host connectivity callback

Returns a key value “host” with ice candidate information as description for internal network.

JS

```
function diagnosticsKeyValueCallback(key, status, description) {
// key:“host”
// key: connected/disconnected
// description : ice candidate for the host connectivity (local facing)/empty string
  }
```

## Reflexive connectivity callback

Returns a key value “srflx” with ice candidate information as description for external network.

JS

```
 function diagnosticsKeyValueCallback(key, status, description) {
// key:“srflx”
// key: connected/disconnected
// description : ice candidate for the reflex connectivity (remote facing)/empty string
  }
```

## Auto Reconnect

Sometimes due to network issues, websocket connections get disconnected. In that case the application has to retry the connection. To implement it we can store the state for shouldAutoRetry, and during doRegistration it could be set as true, and during explicit unregistration it could be set as false, and based on an unregistered event we can invoke doRegister API.

JS

```
var shouldAutoRetry = false;
function registerToggle() {
   if (document.getElementById("registerButton").innerHTML === "REGISTER") {
       shouldAutoRetry = true;
       UserAgentRegistration();
   } else {
       shouldAutoRetry = false;
       exWebClient.unregister();
   }
}
```

JS

```
function RegisterEventCallBack(state, sipInfo) {
   document.getElementById("status").innerHTML = state;
   if (state === 'registered') {
       document.getElementById("registerButton").innerHTML = "UNREGISTER";
   } else {
       document.getElementById("registerButton").innerHTML = "REGISTER";
       if (shouldAutoRetry) {
           exWebClient.DoRegister();
       }
   }
}
```

## Check SDK Readiness

- To check the SDK readiness, whether SDK is ready to receive a call or not. We can invoke checkClientStatus API with a callback method.

- First it checks if the microphone is available or not, then it checks whether the websocket is connected or not, then it checks if the user is registered or not.

| Args | Datatype |
| --- | --- |
| clientStatusCallback | Callback function with status as String |

| Event | Event Decription |
| --- | --- |
| media_permission_denied | either media device not available, or permission not given |
| not_initialized | sdk is not initialized |
| websocket_connection_failed | websocket connection is failing, due to network connectivity |
| unregistered,terminated | either your credential is invalid or registration keepalive failed. |
| initial | sdk registration is progress |
| registered | Ready to receive the calls |
| unknown | something went wrong |
| disconnected | websocket is not connected |
| connecting | Trying to connect the websocket |

JS

```
exWebClient.checkClientStatus(function (status) {
       console.log("SDK Status " + status);
 });
```

## Audio Device Selection

- To get the device ID when the default device got changed, we can register callbacks
- registerAudioDeviceChangeCallback function Argument
  -

| Argument | type | |
| --- | --- | --- |
| audioInputDeviceChangeCallback | function | manadatory |
| audioOutputDeviceCallback | function | mandatory |
| onDeviceChangeCallback | function | optional |

- In case we dont pass onDeviceChangeCallback then sdk will internally try to change the default input/output device
- If onDeviceChangeCallback is passed as third argument then sdk will not try to change the default audio/input device internally, however, OS may have change the default device at OS level.

JS

```
exWebClient.registerAudioDeviceChangeCallback(function (deviceId) {
    console.log(`demo:audioInputDeviceCallback device changed to ${deviceId}`);
}, function (deviceId) {
    console.log(`demo:audioOutputDeviceCallback device changed to ${deviceId}`);
});
```

- During the call or before the call, to change the audio output device
- You can optionally set the `forceDeviceChange` parameter to `true`. This action will bypass the system's internal auto-switching mechanisms.

JS

```
exWebClient.changeAudioOutputDevice(
        selectedDeviceId,
        () => console.log(`Output device changed successfully`),
        (error) => console.log(`Failed to change output device: ${error}`),     true // optional
    );
```

- During the call or before the call, to change the audio input device
- You can optionally set the `forceDeviceChange` parameter to `true`. This action will bypass the system's internal auto-switching mechanisms.

JS

```
function changeAudioInputDevice() {
    const selectedDeviceId = document.getElementById('inputDevices').value;
    exWebClient.changeAudioInputDevice(
        selectedDeviceId,
        () => console.log(`Input device changed successfully`),
        (error) => console.log(`Failed to change input device: ${error}`),
  true // optional
    );
}
```

- If you want the SDK to automatically detect and switch to newly plugged in audio input/output devices, you can enable this option by passing a 4th argument to `initWebrtc`.

JS

```
exWebClient.initWebrtc(
  sipAccountInfo,
  RegisterEventCallBack,
  CallListenerCallback,
  SessionCallback,
  true // optional: Enables auto audio device change handling
);
```

## Noise Suppression

- The SDK provides control over noise suppression for improved call quality. By default, noise suppression is disabled.
- Call after DoRegister().

| API Name | Args |
| --- | --- |
| exWebClient.setNoiseSuppression | Boolean (true / false) |

JS

```
// Enable noise suppression
exWebClient.setNoiseSuppression(true);

// Disable noise suppression
exWebClient.setNoiseSuppression(false);
```

## Logger Callback

To get the SDK logs item as a callback event we can register own logger.
registerLoggerCallback is **static** methods in ExotelWebClient

RegisterLoggerCallback args

| Args | Datatype |
| --- | --- |
| type | string |
| message | String |
| args | Array |

JS

```
exotelSDK.ExotelWebClient.registerLoggerCallback(function (type, message, args) {
    switch (type) {
        case "log":
            console.log(`demo: ${message}`, args);
            break;
        case "info":
            console.info(`demo: ${message}`, args);
            break;
        case "error":
            console.error(`demo: ${message}`, args);
            break;
        case "warn":
            console.warn(`demo: ${message}`, args);
            break;
        default:
            console.log(`demo: ${message}`, args);
            break;
    }
});
```

## Audio Volume Control

- The SDK provides granular control over different audio elements including call audio, ringtone, ringback tone, DTMF tone, and beep tone volumes.
- Volume values are normalized between 0.0 (silent) and 1.0 (maximum)
- Volume settings persist during the session but reset when the page is reloaded
- Since the notifications are global, these functions are static

## Notification Audio Volume Control

- Methods to control audio output volume for notification sounds.

| API Name | Args | Returns | Description |
|---|---|---|---|
| ExotelWebClient.setAudioOutputVolume | - audioElementName (string)<br>- value (number 0.0-1.0) | None | Set notification sound volume |
| ExotelWebClient.getAudioOutputVolume | - audioElementName (string) | Current volume (number 0.0-1.0) | Get notification sound volume |

- Valid audioElementName values:
  - "ringtone" - Incoming call ringtone
  - "ringbacktone" - Outgoing call ringback tone
  - "dtmftone" - DTMF keypad tones
  - "beeptone" - System beep sounds

JS

```
// eg: Set ringtone volume to 50%
exotelSDK.ExotelWebClient.setAudioOutputVolume("ringtone", 0.5);

// Get current ringtone volume
const volume = exotelSDK.ExotelWebClient.getAudioOutputVolume ("ringtone");
```

## Call Audio Volume Control

- Methods to control audio output volume for call audio.
- Since this method is call volume per account, this function is not static

| API Name | Args | Returns | Description |
| --- | --- | --- | --- |
| exWebClient.setCallAudioOutputVolume | - value (number 0.0-1.0) | None | Set call audio volume |
| exWebClient.getCallAudioOutputVolume | - None | Current volume (number 0.0-1.0) | Get call audio volume |

JS

```
// eg: Set call audio volume to 80%
exWebClient.setCallAudioOutputVolume(0.8);

// Get current call audio volume
const callVolume = exWebClient.getCallAudioOutputVolume();
```

## Disabling Built-in Logging

- The SDK provides a way to control all built-in logging (console output, SDK logs, and SIP.js logs) using the setEnableConsoleLogging method.
- setEnableConsoleLogging is **static** methods in ExotelWebClient:

JS

```
// Disable all SDK and SIP.js logs
exotelSDK.ExotelWebClient.setEnableConsoleLogging(false);
```

- Default: Logging is enabled (true).
- Effect: When set to false, all SDK logs, SIP.js logs, and internal logger callbacks are suppressed.
- Note: This should be called before initializing or registering the client to ensure no logs are printed.

## Integration with Exotel APIs

Refer to https://developer.exotel.com/api/make-a-call-api for exotel platform integration APIs. For example, to make a outbound call from a webclient to a pstn phone the request will be

Curl

```
curl -s -X POST https://<your_api_key>:<your_api_token><subdomain>/v1/Accounts/<your_account_sid>/Calls/connect -d "From=<sip_user
```

# 4.2.2.3. SDK Samples

| | | |
|---|---|---|
| SDK | https://github.com/exotel/webrtc-client-sdk | |
| Sample App | https://github.com/exotel/exotel-voip-websdk-sampleapp | |
| Releases | https://github.com/exotel/webrtc-client-sdk/releases | |
| | | |

# 4.2.3. Subscriber Management API

This guide explains how to manage subscribers (end-users or agents) in your application using Exotel's Client SDK. These APIs let you create, update, fetch, and delete subscriber accounts that are used for WebRTC/VoIP calling.

**Base URL**:

```
India: https://api.in.exotel.com/v2/accounts/{accountSid}
SG: https://api.exotel.com/v2/accounts/{accountSid}

Auth: Basic Auth with AccountSid:Token
```

## 1. Create Subscribers

Create one or more subscribers in a single request (up to **25 subscribers** at a time).

**Endpoint:**

`POST /subscribers`

**Request Body Example:**

JSON

```json
{
  "subscribers": [
    {
      "user_name": "alice",
      "password": "********",
      "email": "alice@example.com"
    },
    {
      "user_name": "bob",
      "password": "********",
      "email": "bob@example.com"
    }
  ]
}
```

**Response Example:**

JSON


```
{
  "request_id": "22c70a0c46414af9b7136491c31f95a6",
  "method": "POST",
  "http_code": 200,
  "metadata": {
    "total": 2,
    "success": 2
  },
  "response": [
    {
      "code": 200,
      "status": "success",
      "data": {
        "user_name": "alice",
        "date_created": "2023-06-05T09:32:49Z",
        "date_updated": "2023-06-05T09:32:49Z",
        "email": "alice@example.com",
        "domain": "<accountSid>.voip.exotel.com",
        "status": "inactive",
        "display_name": null,
        "dc_id": null
      }
    },
    {
      "code": 200,
      "status": "success",
      "data": {
        "user_name": "bob",
        "date_created": "2023-06-05T09:32:49Z",
        "date_updated": "2023-06-05T09:32:49Z",
        "email": "bob@example.com",
        "domain": "<accountSid>.voip.exotel.com",
        "status": "inactive",
        "display_name": null,
        "dc_id": null
      }
    }
  ]
}
```


## 2. Get Subscriber Details

Fetch details of subscribers by username or list with pagination.

**Endpoint:**

`GET /subscribers`

**Query Parameters:**

- user_name (optional, comma separated)

- offset (default 0)
- page_size (default 250, max 250)

**Sample Request:**

```
GET /subscribers?user_name=alice,bob&offset=0&page_size=250
```

**Response Example:**

JSON


```json
{
  "request_id": "f293bfb68b6e4a74a0f3bab31946c59b",
  "method": "GET",
  "http_code": 200,
  "metadata": {
    "page_size": 250,
    "first_page_uri": "/v2/accounts/<accountSid>/subscribers?offset=0&page_size=250&user_name=alice,bob",
    "prev_page_uri": null,
    "next_page_uri": null
  },
  "response": [
    {
      "code": 200,
      "status": "success",
      "data": {
        "user_name": "alice",
        "date_created": "2023-06-05T09:32:49Z",
        "date_updated": "2023-06-05T09:32:49Z",
        "email": "alice@example.com",
        "domain": "<accountSid>.voip.exotel.com",
        "status": "inactive",
        "display_name": null,
        "dc_id": null
      }
    },
    {
      "code": 200,
      "status": "success",
      "data": {
        "user_name": "bob",
        "date_created": "2023-06-05T09:32:49Z",
        "date_updated": "2023-06-05T09:32:49Z",
        "email": "bob@example.com",
        "domain": "<accountSid>.voip.exotel.com",
        "status": "inactive",
        "display_name": null,
        "dc_id": null
      }
    }
  ]
}
```


## 3. Update Subscriber

Update details for a single subscriber.

**Endpoint:**

`PUT /subscribers/{user_name}`

**Request Body Example:**

JSON

```
{
  "password": "NewStrongPass@123",
  "email": "alice.updated@example.com"
}
```

**Response Example:**

JSON

```
{
  "request_id": "e6b64e1460ac467a9680a8967ba4b4eb",
  "method": "PUT",
  "http_code": 200,
  "response": {
    "code": 200,
    "status": "success",
    "data": {
      "user_name": "alice",
      "date_created": "2023-06-05T09:32:49Z",
      "date_updated": "2023-06-05T09:55:10Z",
      "email": "alice.updated@example.com",
      "domain": "<accountSid>.voip.exotel.com",
      "status": "inactive",
      "display_name": null,
      "dc_id": null
    }
  }
}
```

## 4. Delete Subscribers

Delete up to **50 subscribers** in a single request.

**Endpoint:**

`DELETE /subscribers?user_name=alice,bob`

**Response Example:**

JSON


```
{
  "request_id": "7c004e095c8e4ae5b32e2244a7a1fef8",
  "method": "DELETE",
  "http_code": 200,
  "metadata": {
    "total": 2,
    "success": 2
  },
  "response": [
    {
      "code": 200,
      "status": "success",
      "data": {
        "user_name": "alice",
        "date_created": "2023-06-05T09:32:49Z",
        "date_updated": "2023-06-05T09:32:49Z",
        "email": "alice@example.com",
        "domain": "<accountSid>.voip.exotel.com",
        "status": "inactive",
        "display_name": null,
        "dc_id": null
      }
    },
    {
      "code": 200,
      "status": "success",
      "data": {
        "user_name": "bob",
        "date_created": "2023-06-05T09:32:49Z",
        "date_updated": "2023-06-05T09:55:10Z",
        "email": "bob@example.com",
        "domain": "<accountSid>.voip.exotel.com",
        "status": "inactive",
        "display_name": null,
        "dc_id": null
      }
    }
  ]
}
```


## Additional Notes

- Subscribers are required for **WebRTC SDK login & registration**.
- Default subscriber status is inactive until first SDK registration.
- Bulk limits: **25 on create**, **50 on delete**.
- Use strong passwords and rotate them periodically.
- Subscriber domain always follows:
  `<accountSid>.voip.exotel.com`

# 4.2.4. Outbound Call APIs

## Integration with Exotel APIs

Refer to https://developer.exotel.com/api/make-a-call-api for exotel platform integration APIs. For example, to make a outbound call from a webclient to a pstn phone the request will be

Curl

```
curl -s -X POST https://<your_api_key>:<your_api_token><subdomain>/v1/Accounts/<your_account_sid>/Calls/connect -d "From=<sip_user
```

## Support Contact

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

# 4.3. Client Android SDK

## 4.3.1. Introduction and Glossary

### 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.

### 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**

### 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 |

# 4.3.1.1. Android SDK Integration

## Getting Started

### 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

### 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

1. Go to File -> Project structure -> Dependencies > “+” that says “All Dependencies > Add path of AAR
2. 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,)'
```

## Target Platform

- Supported Android Version: 10.0+
- Supported Android ABIs: armeabi-v7a, arm64-v8a, x86_64 and x86

## 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" />
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.MANAGE_OWN_CALLS"/>
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT"/>


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

## 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.

## 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);
              }
          }
      });

}
```

## 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(
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 | https://miles.apac-sg.exotel.in/v2 |
| Subscriber Name | 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. |
| Subscriber Token | can be gotten from the API in the `Get Subscriber Token` section in the Subscriber Management API document. In the *subscriberToken*, both the *refresh token* and *access token* are base64 encoded. |

Subscriber token format example

```
"subscriber_token":
{
"refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3Njc0Nzk5OSwiY2xpZW50X2lkIjoiNUUwNTg2NUUifQ.Hc3umVfFlKlPiJ8R9kcP9o9hE9he51le08rO22u7eqs",
"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3Njc0Nzk5OSwic2NvcGUiOiJ2b2IjZSIsImNsaWVudF9pZCI6IjVFMDU4NjVFIn0.uQd_aHGBIT4XqTPfl-567dmjQrKlmLtPag0KpGB2QMA"
}
```

Below are the sample methods to get the subsriberToken, 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.

## 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();
```

## 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 -

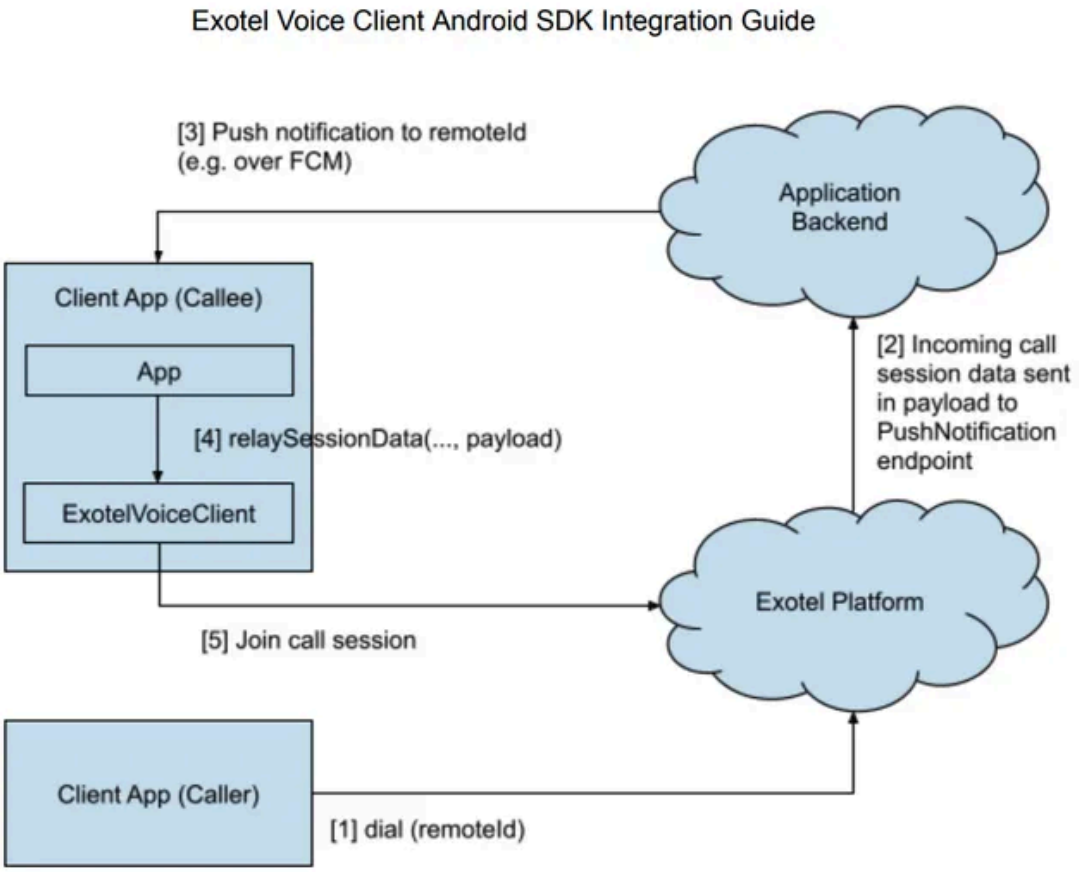
| Param | Mandatory | Description |
|---|---|---|
| sip_exophone | Yes | ExotelVoice library always routes the call via SIP Exophone configured for your account. SIP Exophones are different from the PSTN / Mobile Exophones. They can be identified using sip: prefix. Contact exotel support to enable SIP Exophone in your account.<br><br>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 Dial Whom Endpoint for details. |
| custom_field | 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 - {}()[] |

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

## 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 *ExotleVoiceClient* 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();
```

## Hangup Call

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

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

## 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();
```

## 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.<br>To handle such disruptions effectively ensuring continuity or termination based on severity it initiates reconnection procedures and moves to RECONNECTING if RTP loss persists.<br>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 | - | - |
| 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() | - |

## 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.

## 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.

## 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.

## 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.

## 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 |

# 4.3.1.1.1. Platform Integration

## Platform Integration

### 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.

### 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=743ddcf5a0552050bc37b6d0ff9613bn&CallFrom=sip:Alice&CallTo=sip:08040408080&CallStatus=ringing&Direction=i
ncoming&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&DialWhomNumber=&flow_id=249196&tenant_id=113828&From=sip:Alice&To=sip:08040408080&CurrentTime=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
}
```


## 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.

# Exotel API Endpoints

## 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,

Pre-provisioning

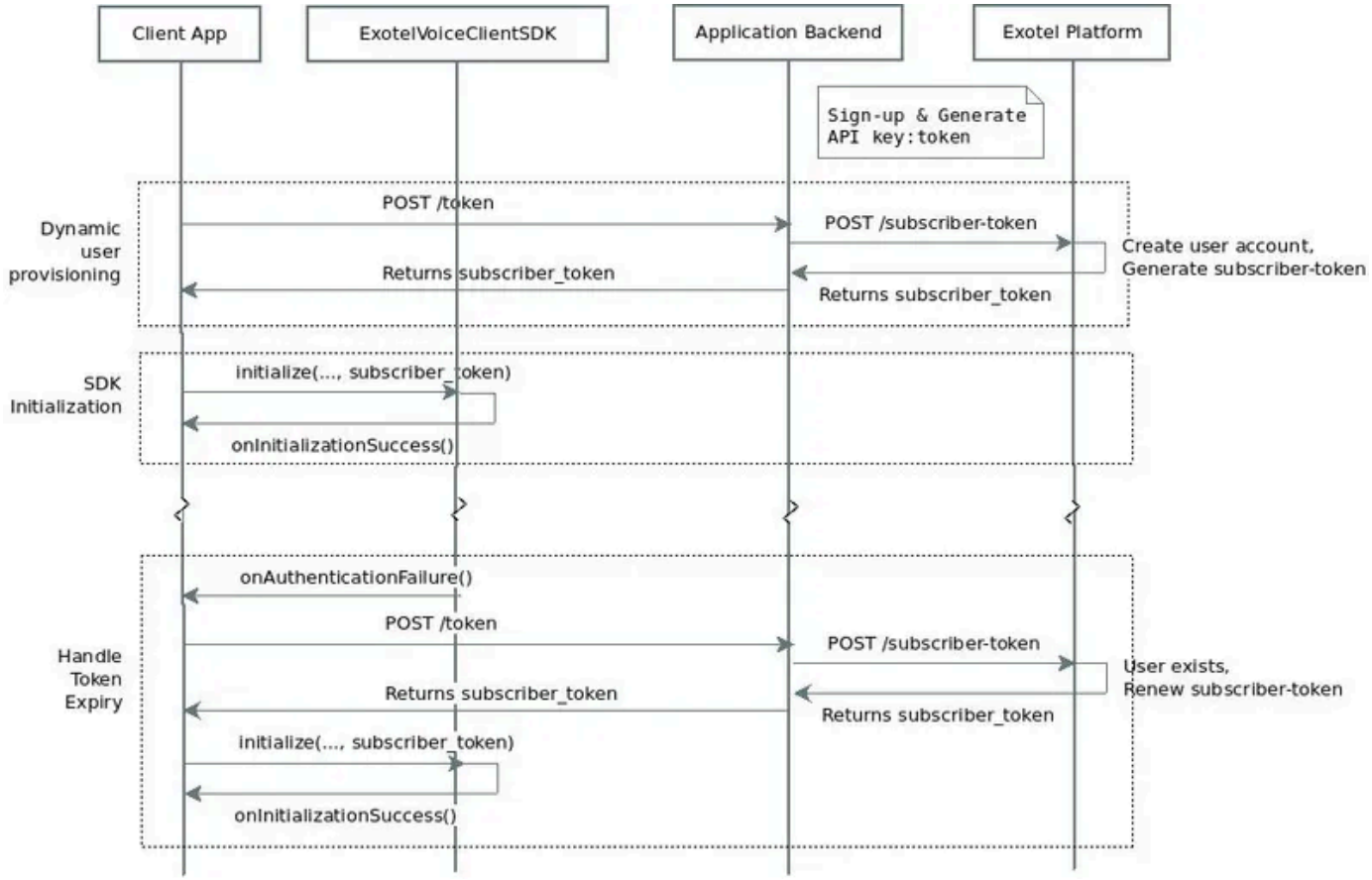
The customer pre-configures the client accounts even before the app is installed by the client. Refer to the *"Subscriber-Management-API"* document for details on creating client accounts.

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.

# 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

Reference Documents

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

Support Contact

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

## Troubleshooting Guide

Outgoing Call

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

1. Check if *"Dial Whom"* URL was configured in *Connect Applet* and reachable.
2. 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 C*allListener::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
3. Report the issue to *exotel support* with *sessionId* (from App) and *Call-Reference-Id* (if available, from account dashboard)

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.

## Incoming Call

My calls are not landing on the callee app?

1. 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.
2. 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.
3. 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()*
4. Report the issue to *exotel support* with *Call-Reference-Id* (from account dashboard)

Problems with push notifications on Xiaomi devices

1. 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.

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.

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.

## Authentication

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. |

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

Check the *ErrorType* in response. If it is,

- AUTHENTICATION_INVALID_TOKEN

Invalid Token.
Initialize new token

- AUTHENTICATION_EXPIRED_TOKEN

Token expired. Initialize new token

Contact **Exotel support** if the issue persists.

# 4.3.1.1.2. SDK Sample

| | |
|---|---|
| Sample SDK | **https://github.com/exotel/exotel-voip-sdk-android** |
| Latest Releases | **https://github.com/exotel/exotel-voip-sdk-android/releases** |

# 4.3.2. Subscriber Management API

## Introduction

This document defines the exotel platform APIs adding, updating, viewing and deleting the subscribers for VoIP Calling. Subscriber is the entity who uses the app with voip calling feature.

Subscriber provisioning in exotel can happen in two ways,

- Pre-provisioning

The customer pre-configures the client accounts even before the app is installed by the client.

- Dynamic provisioning

A client account is created after the app is installed by the user and signs-up with the application backend.

All the APIs listed in this document are privileged APIs that need API Key and API Token for authentication. API key:token are generated on sign-up with exotel platform. Please contact hello@exotel.in

***Base URL***

In all the APIs below, replace <base_url> with the following based on the stamp. Stamp info can be found in your Exotel dashboard under the API settings page.

- MUM Stamp - https://chaotix.mum1.exotel.in
- SGP Stamp - https://chaotix.apac-sg.exotel.in

## Subscriber Response Object

The subscriber object returned in the APIs is described below

| Parameter | Data Type | Description |
|---|---|---|
| subscriber_name | String | Subscriber name provided while creating the subscriber (Must be unique for a particular tenant)<br><br>Maximum supported subscriber name length is 16 bytes and can contain numeric characters. Whitespace characters are not allowed.<br>e.g. 123456, 9889899999 etc |
| status | Enum (active/ inactive) | Whether the subscriber is active or not |
| custom_field | String | Field to store custom metadata for the subscriber |
| date_created | DateTime | Time at which the subscriber was created |
| date_updated | DateTime | Time at which the subscriber was updated |

## Create Subscriber Token

Access to the exotel platform by ExotelVoiceClient is authenticated using a bearer token - *subsriber_token*. Clients must obtain this token from their application backend which in turn should fetch it from the exotel platform using the HTTPS endpoint listed here.

**Method:** `POST`

**Request URL:** `https://<base_url>/v2/accounts/<account-sid>/subscriber-token`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:**

| Parameter | Data Type | Description |
|---|---|---|
| platform | String | Parameter (Android / iOS) |
| device_id | String | Device id (AndroidID / iOS UDID) |
| subscriber_name | String | Unique subscriber identifier |

**Response Type:** `JSON`

**Response Code:**

`200 Success`

`400 Bad Request, Malformed Parameters`

`401 Unauthorized`

**Response Body Parameters**:

| Parameter | Data Type | Description |
|---|---|---|
| subscriber_token | String | Token to access exotel platform |
| subscriber_name | String | Subscriber name used for token generation |

**Note** - The Time-to-Live (TTL) for token expiry is as follows:

Refresh token expiry: 24 hours

Access token expiry: 30 minutes

Sample Request

JS

```
https://<base_url>/v2/accounts/exotel13m2/subscriber-token
{
"platform": "android",
"device_id": "2fc4b5912826ad1",
"subscriber_name": “archit”
}
```

Sample Response

JSON


```
{
"request_id": "2f97c5e98e1d4c188c483cbfe8092869",
"method": "POST",
"http_code": 200,
“response”:
{
"code": 200,
"error_data": null,
"status": "success",
“data”:
{
    "subscriber_name": “archit”,
"subscriber_token":
{
"refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3
"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3N
}
},
}
}
```


## Create Subscribers

API is used for both single or bulk subscriber account creation in exotel platform.

**Method:** `POST`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters**:

| Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| subscriber_name | String | Mandatory | Unique Username for the Subscriber |
| custom_field | String | Optional | Field to store custom metadata for the Subscriber |

Please note that POST is a bulk operation and the request is actually an array of the above.

The maximum number of subscribers which can be created in a single request is 10.

**Response Type:** `JSON`

**Response Body Parameters:** Returns multipart response for request. Refer Subscriber Response Object

Sample Request

JS

```
https:///<base_url>/v2/accounts/<accountsid>/subscribers
{
  "subscribers": [
    {
      "subscriber_name": "bob",
      "custom_field": "support"
    },
    {
      "subscriber_name": "nidhi"
    },
    {
      "subscriber_name": "archit"
    }
  ]
}
```

Sample Response

JSON


```
{
"request_id": "04386e350256448dae83c6db0f749a21",
"method": "POST",
"http_code": 207,
"metadata": {
"failed": 0,
"success": 3
},
"response": [
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:17+05:30",
"date_updated": "2019-11-19T23:14:17+05:30",
"subscriber_name": "bob",
"status": "active",
"custom_field": "support”
}
},
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:17+05:30",
"date_updated": "2019-11-19T23:14:17+05:30",
"subscriber_name": "nidhi",
"status": "active",
"custom_field": null
}
},
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-19T23:14:18+05:30",
"subscriber_name": "archit",
"status": "active",
"custom_field": null
}
}
],
}
```


## Update a Subscriber

API is used to update the subscriber account parameters after it has been created. Subscriber can be marked inactive for temporary suspension.

**Method:** `PUT`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber-name>`

**Authorization Type:** Basic authentication using API key:token

**Request Body Parameters**:

| Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| custom_field | String | Optional | Field to store custom metadata for the subscriber |
| status | Enum | Mandatory | Indicates whether the subscriber is active or not<br>enum {"active", "inactive"} |

**Response Type:** JSON

**Response Body Parameters:** Refer Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/archit
{
  "status": "inactive",
}
```

Sample Response

JSON

```
{
"request_id": "8019271c8d98422db509c2801e63435f",
"method": "PUT",
"http_code": 200,
"response": {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-20T11:13:38+05:30",
"subscriber_name": "archit",
"status": "inactive",
"custom_field": null
}
}
}
```

## Get Subscriber

API returns subscriber details.

**Method:** `GET`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber_name>`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:** None

**Response Type:** `JSON`

**Response Body Parameters:** On success returns Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/archit
```

Sample Response

JSON


```
{
"request_id": "287c227674af40b38531a1c247262deb",
"method": "GET",
"http_code": 200,
"response": {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-20T11:13:38+05:30",
"subscriber_name": "archit",
"status": "inactive",
"custom_field": null
}
}
}
```


## List Subscribers

API returns a paginated list of subscribers.

**Method:** `GET`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers>?status=abc&page_size=xyz&offset=xyz`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:**

| Query Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| status | Enum | Optional | Status for the subscriber |
| page_size | uint64 | Optional | Page size for response. Default and maximum value is 100 |
| offset | uint64 | Optional | Offset from which to get the records. Defaults to 0 |

**Response Type:** `JSON`

**Response Body Parameters:** On success returns paginated list of Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers?page=2&offset=3
```

Sample Response

JSON


```
{
"request_id": "b5f0c6780ec443d08947ba2015a06a04",
"method": "GET",
"http_code": 200,
"metadata": {
"prev_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=6&page_size=3",
"next_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=12&page_size=3",
"first_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=0&page_size=3"
},
"response": [
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T17:12:00+05:30",
"date_updated": "2019-12-30T17:12:00+05:30",
"subscriber_name": "shyam",
"status": "active",
"custom_field": null
}
},
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T17:09:41+05:30",
"date_updated": "2019-12-30T17:09:41+05:30",
"subscriber_name": "kiran",
"status": "active",
"custom_field": "test"
}
},
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T13:35:07+05:30",
"date_updated": "2019-12-30T13:35:07+05:30",
"subscriber_name": "mandeep",
"status": "active",
"custom_field": null
}
}
],
}
```


## Delete Subscriber

API deletes a subscriber permanently.

**Method:** `DELETE`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber-name>`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:** None

**Response Type:** `JSON`

**Response Body Parameters:** On success, HTTP response 204 is returned with no response body. In case of failure 404 is returned

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/shyam
```

Sample Response

(a) Success: `No body`

(b) Failure:

JSON


```
{
    "request_id": "af0087d5331f479599d2c987efe9f664",
    "method": "DELETE",
    "http_code": 404,
    "response": {
        "code": 404,
        "error_data": {
            "code": 1000,
            "description": "Subscriber not found",
            "message": "Not Found"
        },
        "status": "failure",
        "data": null
    }
}
```

# 4.4. Client IOS SDK

## 4.4.1. Integration Guide

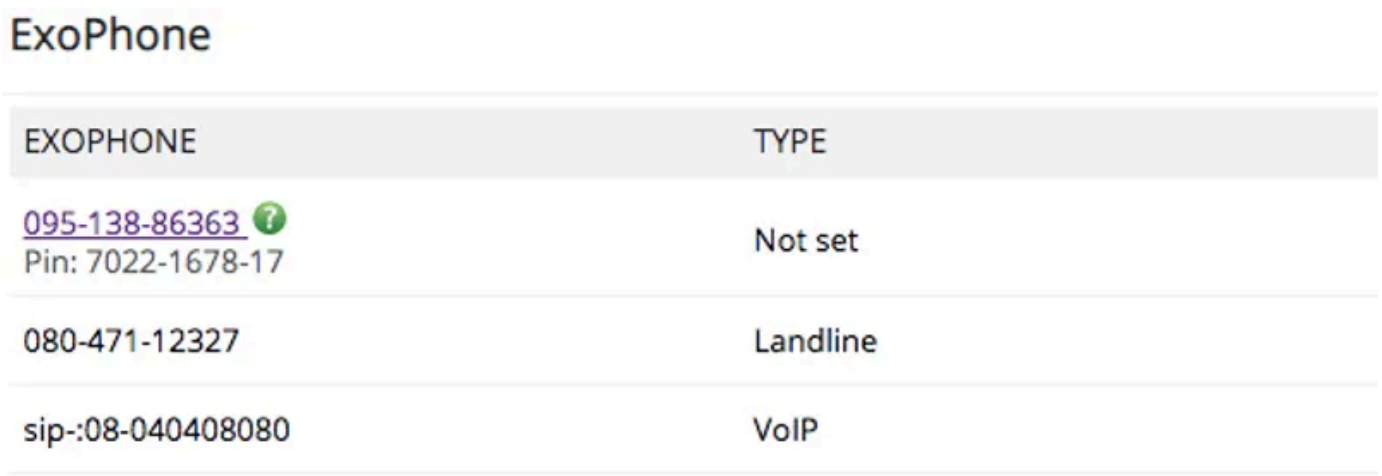
### Introduction

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.

### 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 .




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**

### 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 |

# IOS SDK Integration

## Getting Started

## 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

## Add Exotel Voice SDK Library to Project

Configure ExotelVoice.Framework

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

## Target Platform

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

## Permissions

- Record permission
- Notification permission

Proguard Rules - Not applicable

## 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.

# 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 = 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 |
| Subscriber Name | 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. |
| Subscriber Token | can be gotten from the API in the `Get Subscriber Token` section in the Subscriber Management API document. In the *subscriberToken*, both the *refresh token* and *access token* are base64 encoded. |

Subscriber token format example

```
"subscriber_token":
{
"refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3Njc0Nzk5OSwiY2xpZW50X2lkIjoiNUUwNTg2NUUifQ.Hc3umVfFlKlPiJ8R9kcP9o9hE9he51le08rO22u7eqs",
"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3Njc0Nzk5OSwic2NvcGUiOiJ2b2IjZSIsImNsaWVudF9pZCI6IjVFMDU4NjVFIn0.uQd_aHGBIT4XqTPfl-567dmjQrKImLtPag0KpGB2QMA"
}
```

Below are the sample methods to get the subsriberToken, 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.
- 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.

## 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()

## 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()
```

## 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 -**

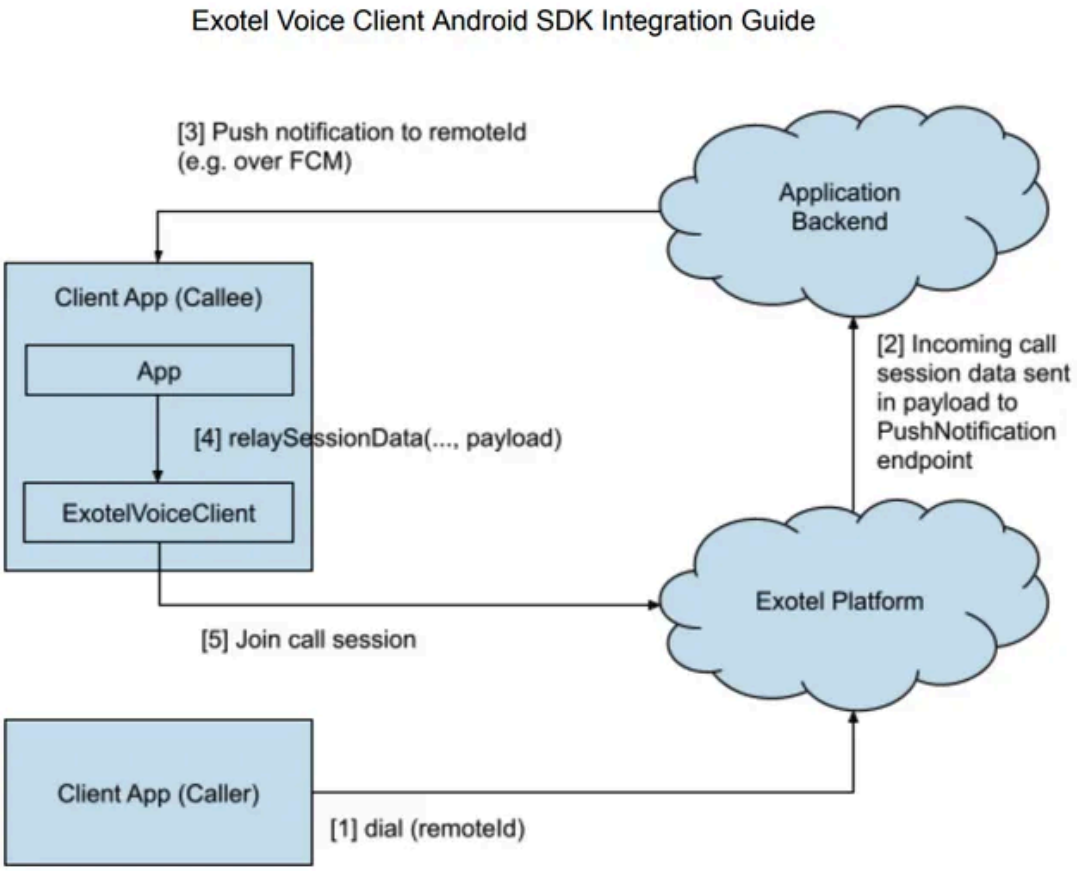
| Param | Mandatory | Description |
|---|---|---|
| remotelD | Yes | ExotelVoice library always routes the call via SIP Exophone configured for your account. SIP Exophones are different from the PSTN / Mobile Exophones. They can be identified using sip: prefix. Contact exotel support to enable SIP Exophone in your account.<br><br>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 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 - {}()[] |

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()
```

## 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
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()
```

## 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

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

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

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)
```

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

```
func provider(_ provider: CXProvider, perform action: CXStartCallAction) {

    setActiveUUID(uuid: action.callUUID)

    action.fulfill()

}
```

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)
```

Timer will start in call kit UI.

## Incoming Calls

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.

## 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:CXAnswerAllAction)*** 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()
   }
}
```

## 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()
```

## 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.<br>To handle such disruptions effectively ensuring continuity or termination based on severity it initiates reconnection procedures and moves to RECONNECTING if RTP loss persists.<br>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 | - | - |

## 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.

## 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.

## 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.

Handling of PSTN calls when Call Kit is Integrated

*ExotelVoice* 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 the 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.

## 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: startDate, endDate: endDate, description: description)
```

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

## 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 |

# Platform Integration

## 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.

## 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=743ddcf5a0552050bc37b6d0ff9613bn&CallFrom=sip:Alice&CallTo=sip:08040408080&CallStatus=ringing&Direction=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&DialWhomNumber=&flow_id=249196&tenant_id=113828&From=sip:Alice&To=sip:08040408080&CurrentTime=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
}
```


## 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.

# Exotel API Endpoints

## 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,

Pre-provisioning

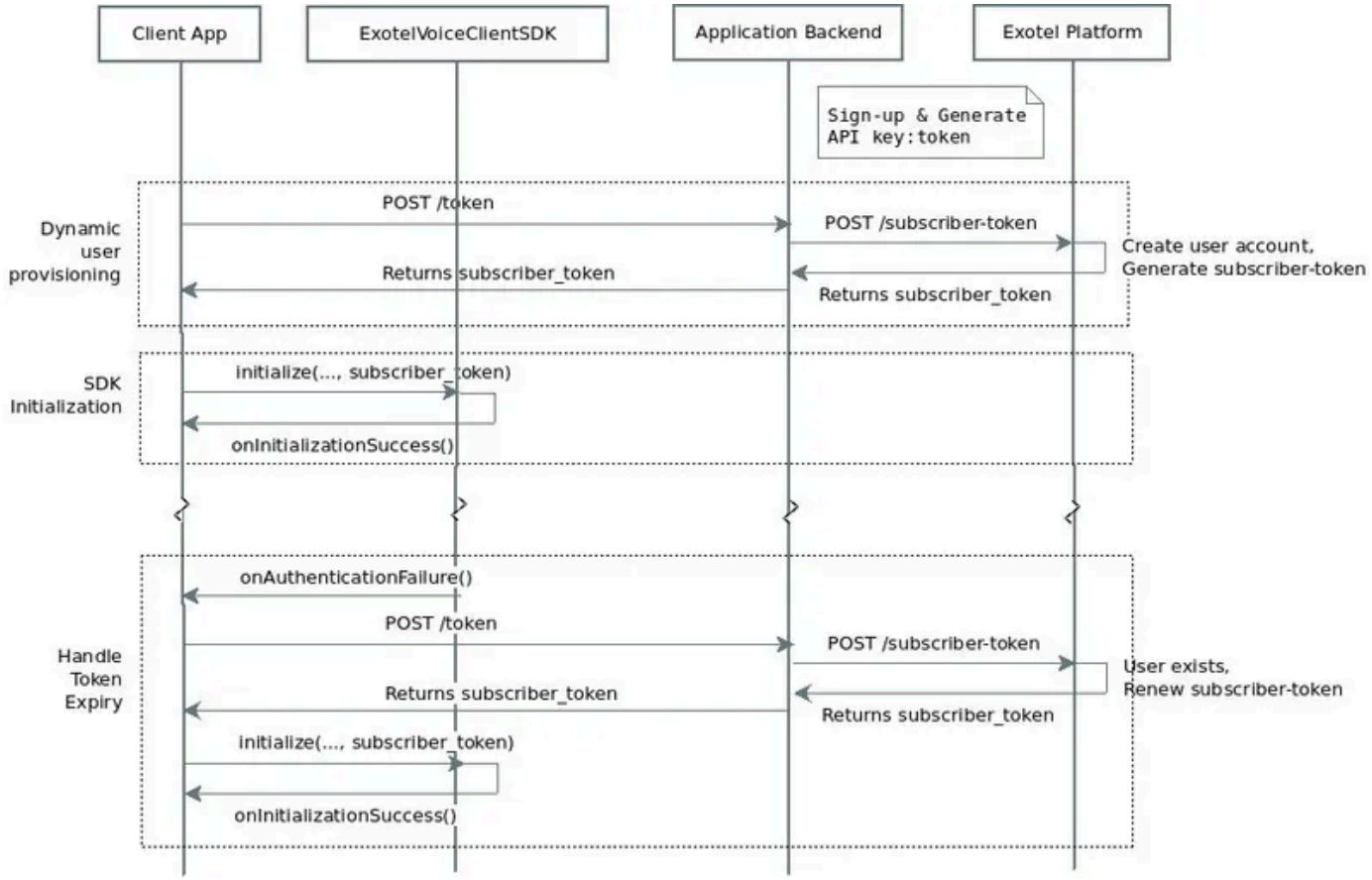
The customer pre-configures the client accounts even before the app is installed by the client. Refer to the *"Subscriber-Management-API"* document for details on creating client accounts.

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.

## Authentication and Authorisation

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

Reference Documents

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

### Support Contact

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

## Troubleshooting Guide

### Outgoing Call

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

1. Check if *"Dial Whom"* URL was configured in *Connect Applet* and reachable.
2. 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 C*allListener::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
3. Report the issue to *exotel support* with *sessionId* (from App) and *Call-Reference-Id* (if available, from account dashboard)

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.

## Incoming Call

My calls are not landing on the callee app?

1. 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.
2. 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.
3. 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()*
4. Report the issue to *exotel support* with *Call-Reference-Id* (from account dashboard)

## Authentication

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. |

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

1. Check the *ErrorType* in response. If it is,

- AUTHENTICATION_INVALID_TOKEN

Invalid Token.
Initialize new token

- AUTHENTICATION_EXPIRED_TOKEN

Token expired. Initialize new token

Contact ***Exotel support*** if the issue persists.

# 4.4.2. SDK Sample

| | |
|---|---|
| Sample SDK | **https://github.com/exotel/exotel-voip-sdk-ios** |
| Latest Releases | **https://github.com/exotel/exotel-voip-sdk-ios/releases** |

# 4.4.3. Subscriber Management API

## Introduction

This document defines the exotel platform APIs adding, updating, viewing and deleting the subscribers for VoIP Calling. Subscriber is the entity who uses the app with voip calling feature.

Subscriber provisioning in exotel can happen in two ways,

- Pre-provisioning

The customer pre-configures the client accounts even before the app is installed by the client.

- Dynamic provisioning

A client account is created after the app is installed by the user and signs-up with the application backend.

All the APIs listed in this document are privileged APIs that need API Key and API Token for authentication. API key:token are generated on sign-up with exotel platform. Please contact hello@exotel.in

***Base URL***

In all the APIs below, replace <base_url> with the following based on the stamp. Stamp info can be found in your Exotel dashboard under the API settings page.

- MUM Stamp - https://chaotix.mum1.exotel.in
- SGP Stamp - https://chaotix.apac-sg.exotel.in

## Subscriber Response Object

The subscriber object returned in the APIs is described below

| Parameter | Data Type | Description |
|---|---|---|
| subscriber_name | String | Subscriber name provided while creating the subscriber (Must be unique for a particular tenant)<br><br>Maximum supported subscriber name length is 16 bytes and can contain numeric characters. Whitespace characters are not allowed.<br>e.g. 123456, 9889899999 etc |
| status | Enum (active/ inactive) | Whether the subscriber is active or not |
| custom_field | String | Field to store custom metadata for the subscriber |
| date_created | DateTime | Time at which the subscriber was created |
| date_updated | DateTime | Time at which the subscriber was updated |

## Create Subscriber Token

Access to the exotel platform by ExotelVoiceClient is authenticated using a bearer token - *subsriber_token*. Clients must obtain this token from their application backend which in turn should fetch it from the exotel platform using the HTTPS endpoint listed here.

**Method:** `POST`

**Request URL:** `https://<base_url>/v2/accounts/<account-sid>/subscriber-token`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:**

| Parameter | Data Type | Description |
|---|---|---|
| platform | String | Parameter (Android / iOS) |
| device_id | String | Device id (AndroidID / iOS UDID) |
| subscriber_name | String | Unique subscriber identifier |

**Response Type:** `JSON`

**Response Code:**

`200 Success`

`400 Bad Request, Malformed Parameters`

`401 Unauthorized`

**Response Body Parameters**:

| Parameter | Data Type | Description |
|---|---|---|
| subscriber_token | String | Token to access exotel platform |
| subscriber_name | String | Subscriber name used for token generation |

**Note** - The Time-to-Live (TTL) for token expiry is as follows:

Refresh token expiry: 24 hours

Access token expiry: 30 minutes

Sample Request

JS

```
https://<base_url>/v2/accounts/exotel13m2/subscriber-token
{
"platform": "android",
"device_id": "2fc4b5912826ad1",
"subscriber_name": “archit”
}
```

Sample Response

JSON

```
{
"request_id": "2f97c5e98e1d4c188c483cbfe8092869",
"method": "POST",
"http_code": 200,
“response”:
{
"code": 200,
"error_data": null,
"status": "success",
“data”:
{
    "subscriber_name": “archit”,
"subscriber_token":
{
"refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3
"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3N
}
},
}
}
```

## Create Subscribers

API is used for both single or bulk subscriber account creation in exotel platform.

**Method:** `POST`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters**:

| Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| subscriber_name | String | Mandatory | Unique Username for the Subscriber |
| custom_field | String | Optional | Field to store custom metadata for the Subscriber |

Please note that POST is a bulk operation and the request is actually an array of the above.

The maximum number of subscribers which can be created in a single request is 10.

**Response Type:** `JSON`

**Response Body Parameters:** Returns multipart response for request. Refer Subscriber Response Object

Sample Request

JS

```
https:///<base_url>/v2/accounts/<accountsid>/subscribers
{
  "subscribers": [
    {
      "subscriber_name": "bob",
      "custom_field": "support"
    },
    {
      "subscriber_name": "nidhi"
    },
    {
      "subscriber_name": "archit"
    }
  ]
}
```

Sample Response

JSON


```
{
"request_id": "04386e350256448dae83c6db0f749a21",
"method": "POST",
"http_code": 207,
"metadata": {
"failed": 0,
"success": 3
},
"response": [
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:17+05:30",
"date_updated": "2019-11-19T23:14:17+05:30",
"subscriber_name": "bob",
"status": "active",
"custom_field": "support”
}
},
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:17+05:30",
"date_updated": "2019-11-19T23:14:17+05:30",
"subscriber_name": "nidhi",
"status": "active",
"custom_field": null
}
},
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-19T23:14:18+05:30",
"subscriber_name": "archit",
"status": "active",
"custom_field": null
}
}
],
}
```


## Update a Subscriber

API is used to update the subscriber account parameters after it has been created. Subscriber can be marked inactive for temporary suspension.

**Method:** `PUT`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber-name>`

**Authorization Type:** Basic authentication using API key:token

**Request Body Parameters**:

| Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| custom_field | String | Optional | Field to store custom metadata for the subscriber |
| status | Enum | Mandatory | Indicates whether the subscriber is active or not<br>enum {"active", "inactive"} |

**Response Type:** JSON

**Response Body Parameters:** Refer Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/archit
{
  "status": "inactive",
}
```

Sample Response

JSON

```
{
"request_id": "8019271c8d98422db509c2801e63435f",
"method": "PUT",
"http_code": 200,
"response": {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-20T11:13:38+05:30",
"subscriber_name": "archit",
"status": "inactive",
"custom_field": null
}
}
}
```

## Get Subscriber

API returns subscriber details.

**Method:** `GET`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber_name>`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:** None

**Response Type:** `JSON`

**Response Body Parameters:** On success returns Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/archit
```

Sample Response

JSON


```
{
"request_id": "287c227674af40b38531a1c247262deb",
"method": "GET",
"http_code": 200,
"response": {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-20T11:13:38+05:30",
"subscriber_name": "archit",
"status": "inactive",
"custom_field": null
}
}
}
```


## List Subscribers

API returns a paginated list of subscribers.

**Method:** `GET`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers>?status=abc&page_size=xyz&offset=xyz`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:**

| Query Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| status | Enum | Optional | Status for the subscriber |
| page_size | uint64 | Optional | Page size for response. Default and maximum value is 100 |
| offset | uint64 | Optional | Offset from which to get the records. Defaults to 0 |

**Response Type:** `JSON`

**Response Body Parameters:** On success returns paginated list of Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers?page=2&offset=3
```

Sample Response

JSON


```
{
"request_id": "b5f0c6780ec443d08947ba2015a06a04",
"method": "GET",
"http_code": 200,
"metadata": {
"prev_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=6&page_size=3",
"next_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=12&page_size=3",
"first_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=0&page_size=3"
},
"response": [
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T17:12:00+05:30",
"date_updated": "2019-12-30T17:12:00+05:30",
"subscriber_name": "shyam",
"status": "active",
"custom_field": null
}
},
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T17:09:41+05:30",
"date_updated": "2019-12-30T17:09:41+05:30",
"subscriber_name": "kiran",
"status": "active",
"custom_field": "test"
}
},
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T13:35:07+05:30",
"date_updated": "2019-12-30T13:35:07+05:30",
"subscriber_name": "mandeep",
"status": "active",
"custom_field": null
}
}
],
}
```


## Delete Subscriber

API deletes a subscriber permanently.

**Method:** `DELETE`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber-name>`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:** None

**Response Type:** `JSON`

**Response Body Parameters:** On success, HTTP response 204 is returned with no response body. In case of failure 404 is returned

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/shyam
```

Sample Response

(a) Success: `No body`

(b) Failure:

JSON

```
{
    "request_id": "af0087d5331f479599d2c987efe9f664",
    "method": "DELETE",
    "http_code": 404,
    "response": {
        "code": 404,
        "error_data": {
            "code": 1000,
            "description": "Subscriber not found",
            "message": "Not Found"
        },
        "status": "failure",
        "data": null
    }
}
```

# 4.5. Client Flutter SDK

## 4.5.1. Integration Guide

### Introduction

ExotelVoice flutter SDK 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.

### 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

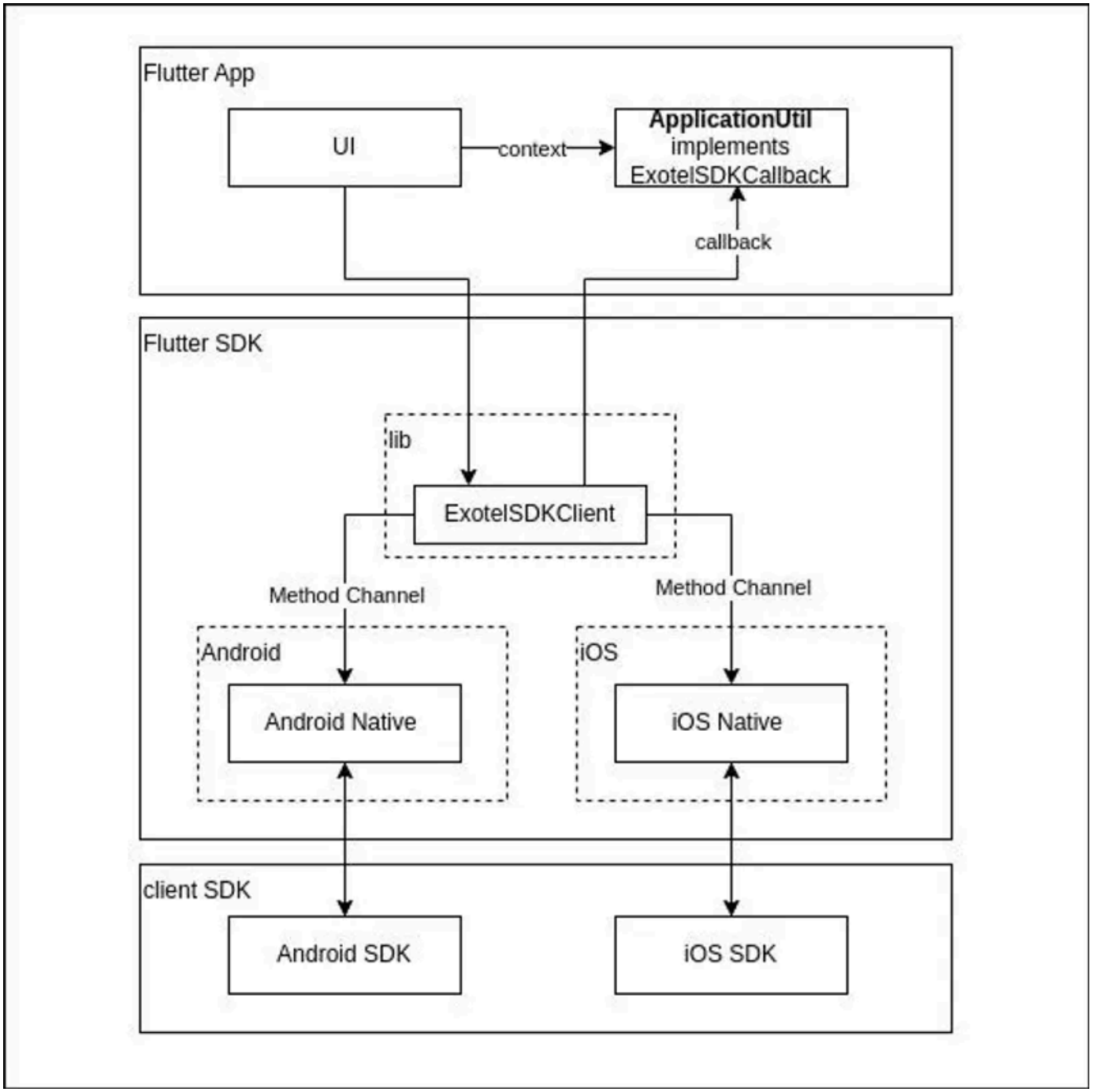
| EXOPHONE | TYPE |
|---|---|
| 095-138-86363<br>Pin: 7022-1678-17 | Not set |
| 080-471-12327 | Landline |
| sip-:08-040408080 | VoIP |

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**

### Flutter SDK Integration

#### SDK Architecture




## Getting Started

**Software Package**

- SDK (tar.gz file) includes android , ios, lib directories
- integration guide
- sample app for reference

**Add SDK to your project**

1. download the SDK from GitHub - **exotel/exotel-voip-sdk-flutter**
2. untar SDK

`tar -xzvf flutter-sdk.tar.gz`

3. copy the following directories in your flutter app
    - `flutter-sdk/android`
    - `flutter-sdk/ios`
    - `flutter-sdk/lib`
4. add native sdk
    - to `android` directory
        - go to android directory and run

          `make deps`

          it will download and add the aar file(exotel-voice-release.aar) at <flutter path>/android/exotel-voice-sdk/
          **Note: you might need to change** EXOTEL_SDK_VERSION **in Makefile to get latest version**
    - to ios directory
        - go to `ios` directory and run

          `make deps`

          it will download and add the aar file(ExotelVoice.xcframework) at <flutter path>/ios/
          **Note: you might need to change** EXOTEL_SDK_VERSION **in Makefile to get latest version**

**SDK File structure**

After extracting the sdk , it contains the following directories

- `android` : android native code that has integrated exotel android sdk.
    - `app/src/main/java/com/exotel/voice_sample`
        - `MainActivity.java` : register the flutter engine and create instance of ExotelSDKChannel.
        - `ExotelSDKChannel.java` : it is android channel interface class which communicate with flutter via Method Channel.
        - `VoiceAppService.java` : this class is communicating with android sdk as per **Android SDK Integration**.
- `ios` : ios native code that has integrated exotel ios sdk.
    - `Runner/AppDelegate.swift` : register the flutter engine and create instance of ExotelSDKChannel
    - `Runner/Translator`
        - ExotelSDKChannel.swift :it is ios channel interface class which communicate with flutter via Method Channel. it is also communicating with ios sdk as per ios **Integration Guide**.
- `lib` : contains flutter dart files
    - `exotelSDK`
        - ExotelVoiceClientFactory.dart: it is factory class of SDK which provide ExotelVoiceClient using method getExotelVoiceClient().
        - ExotelVoiceClient.java: it Exotel Voice Client interface which exposes SDK APIs.

```
abstract class ExotelVoiceClient {
void setExotelSDKCallback(ExotelSDKCallback callback);
void registerPlatformChannel();
```

```
Future getDeviceId();
Future initialize(String hostname, String subsriberName, String displayName, String accountSid,String
subscriberToken);
Future reset();
Future stop();
Future dial(String dialTo, String message);
Future mute();
Future unmute();
Future enableSpeaker();
Future disableSpeaker();
Future enableBluetooth();
Future disableBluetooth();
Future hangup();
Future answer();
Future sendDtmf(String digit);
Future postFeedback(int? rating, String? issue);
Future getVersionDetails();
Future uploadLogs(DateTime startDate, DateTime endDate, String description);
void relaySessionData(Map<String, dynamic> data) {}
}
```

- `ExotelSDKClient.dart` : it is SDK's implementation class which has implemented `ExotelVoiceClient`. it is also flutter channel interface class which communicate with android and ios via Method Channel.
- `ExotelSDKCallaback.dart` : it is SDK Callback or Observer interface that must be implement by your flutter app to handle callbacks from ExotelSDKClient.

```
abstract class ExotelSDKCallback {
  void onInitializationSuccess();
  void onInitializationFailure(String loginStatus);
  void onDeinitialized();
  void onAuthenticationFailure(String loginStatus);
  void onCallInitiated();
  void onCallRinging();
  void onCallEstablished();
  void onCallEnded();
  void onMissedCall();
  void onMediaDisrupted();
  void onRenewingMedia();
  void onCallIncoming(String callId, String destination);
}
```

- `MethodChannelInvokeMethod.dart` : it contains the Channel Method strings which is used to invoke native method of android/ios from flutter.

- Service
  - `PushNotificationService.dart` : sample push notification class.

**Dependency**

- add following dependecny in pubspec.yaml of your flutter project
  - Add permission handling dependency.

    `permission_handler: ^11.3.0`
  - Add firebase dependency.

    `firebase_messaging: ^14.7.20`

**Permissions**

- flutter :

  how to request permission

```
static Future requestPermissions() async {
// You can request multiple permissions at once
Map<Permission, PermissionStatus> statuses = await [
Permission.phone,
Permission.microphone,
Permission.notification,
Permission.nearbyWifiDevices,
Permission.accessMediaLocation,
Permission.location,  Permission.bluetoothScan,
Permission.bluetoothConnect,
// Add other permissions you want to request
].request();
// Check permission status and handle accordingly
}
```

- Android

  android native code must add following permission in `AndroidManifest.xml`

```
<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.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE_PHONE_CALL"/>
<uses-permission android:name="android.permission.MANAGE_OWN_CALLS"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
```

- iOS

  add following depedencies in your `Info.plist`

```
<key>UIBackgroundModes</key>
<array>
<string>audio</string>
<string>fetch</string>
<string>remote-notification</string>
<string>voip</string>
</array>
<key>NSMicrophoneUsageDescription</key>
<string>Enable microphone access to allow other people to hear you.</string>
```

The permission_handler plugin use macros to control whether a permission is enabled.

You must list the permission you want to use in your application:

Add the following to your `Podfile` file:

```
post_install do |installer|
  installer.pods_project.targets.each do |target|
    flutter_additional_ios_build_settings(target)
    target.build_configurations.each do |config|
          config.build_settings['GCC_PREPROCESSOR_DEFINITIONS'] ||= [
            '$(inherited)',

            ## dart: PermissionGroup.microphone
            'PERMISSION_MICROPHONE=1',
          ]
        end
  end
en
```

please refer **permission_handler | Flutter package** to know in detail about permission handling.

## SDK Initialization

Exotel Flutter SDK provide a interface class `ExotelVoiceClient` that exposes `initialize` api to inialize the SDK(Android/iOS).

```
// get ExotelVoiceClient object from Factory class.
ExotelVoiceClient exotelVoiceClient = ExotelVoiceClientFactory.getExotelVoiceClient();

var mApplicationUtil =  ApplicationUtils.getInstance(context); // implemented ExotelSDKCallback
exotelVoiceClient.setExotelSDKCallback(mApplicationUtil);
exotelVoiceClient.registerPlatformChannel();
```

```
//initialization
try {
  exotelVoiceClient.initialize(hostname!, subscriberName!, displayName!,mAccountSid!, subscriberToken!)
} catch (e) {
  log("Error while login");
}
```

| 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 *subscriberToken*, both the *refresh token* and *access token* are base64 encoded.<br>Subscriber token format example:<br>{<br>"refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3Njc0Nzk5OSwiY2xpZW50X2lkIjoiNUUwNTg2NUUifQ.Hc3umVfFlKlPiJ8R9kcP9o9hE9he51le08rO22u7eqs","access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3Njc0Nzk5OSwic2NvcGUiOiJ2b2ljZSIsImNsaWVudF9pZCI6IjVFMDU4NjVFIn0.uQd_aHGBIT4XqTPfI-567dmjQrKlmLtPag0KpGB2QMA"} |

**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.

```
String deviceId = exotelVoiceClient.getDeviceId();
```

In Above code,

1. first you need to get the ExotelSDKClient instance
2. register the Method Handler to handle the callback from android/iOS native module
3. set the callback so that ExotelSDKClient can send callback to UI for some events like *onLoggedInSuccess*, *onLoggedInFailure.* Here it is assumed that ApplicationUtil must implement `ExotelSDKCallback`
4. call the login api with required fields
   userId : sub

Internal working for `initialize` api

- invoking Method "`initialize`" via MethodChannel with required arguments
- on Method "`initialize`" , Android/iOS native method handler will initialize the android/iOS library as mentioned in **Android SDK Integration**

When login is successful,

- client SDK will send callback `onInitializationSuccess()`
- Android/iOS native method handler will then invoke flutter Method `"inialize-result"`
- on flutter Method `"inialize-result"`, ExotelSDKClient will check status and then send `onInitializationSuccess()` callback.

When login is fail,

- client SDK will send callback onInitializationFailure()/onAuthenticationFailure()
- Android/iOS native method handler will then invoke flutter Method `"inialize-result"`
- on flutter Method `"inialize-result"`, `ExotelSDKClient` will check status and then send `onInitializationFailure()/onAuthenticationFailure()` callback.

**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()`
- Android/iOS native method handler will then invoke flutter Method `"on-deinitialized"`
- on flutter Method `"on-deinitialized"`, `ExotelSDKClient` will check status and then send `on-deinitialized()` callback

## Managing Calls

After successful initialization, the library is ready to handle calls. Dialing-out calls or receiving incoming calls is managed through same class ExotelSDKClient.

### Make Outbound Calls

To make outgoing calls, Provide from username and dial to number in dial() API of ExotelSDKClient Interface to make outgoing calls.

`exotelVoiceClient.dial(dialTo,message);`

| Param | Mandatory | Description |
|---|---|---|
| `dialTo` | Yes | 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.<br>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 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 - {}()[] |

In Above Code,

1. `ExotelSDKClient` will invoke "`dial`" method
2. on Method "`dial`" , Android/iOS native method handler will dial the exophone and set the destination number as call context of current user.

When destination user is getting dialed,

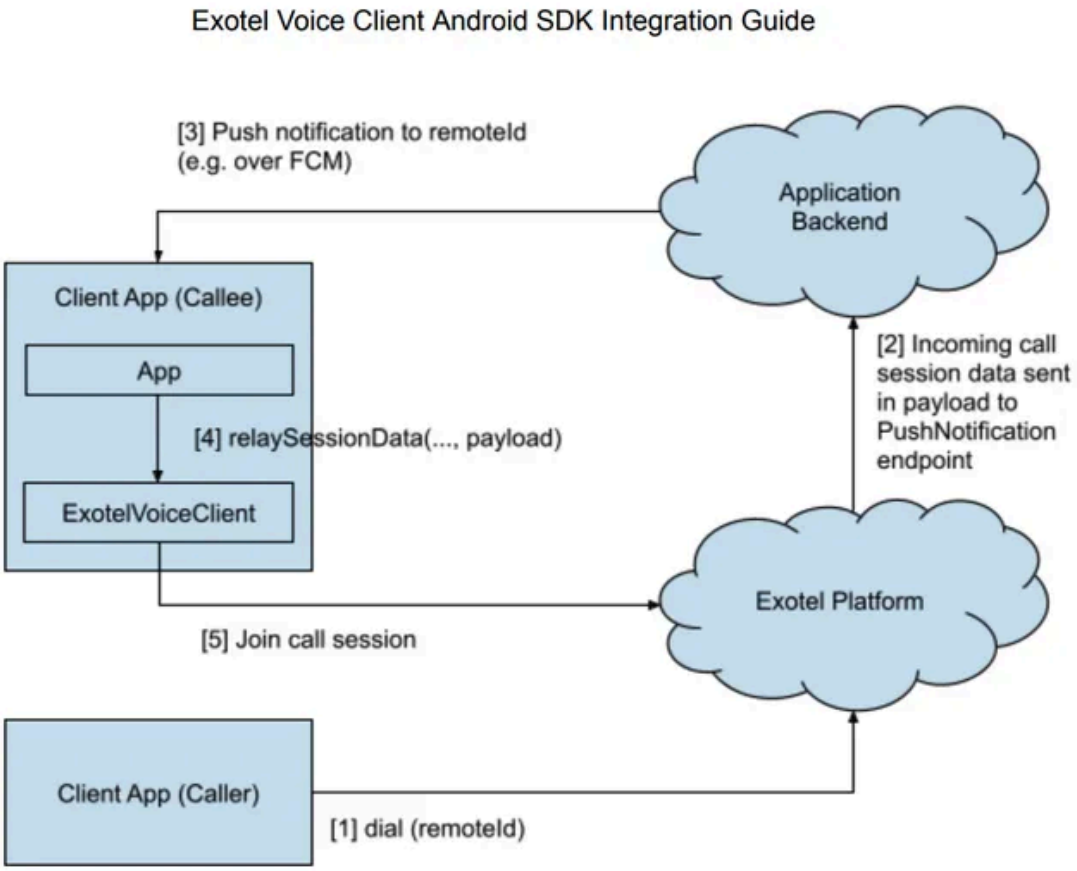
- client SDK will send callback `onCallRinging()`
- Android/iOS native method handler will then invoke flutter Method `"on-call-ringing"`
- on flutter Method `"on-call-ringing"`, `ExotelSDKClient` will check status and then send `onCallRinging()` callback.

When destination user has accepted the call,

- client SDK will send callback `onCallEstablished()`
- Android/iOS native method handler will then invoke flutter Method `"on-call-established"`
- on flutter Method `"on-call-established"`, `ExotelSDKClient` will check status and then send `onCallEstablished()` callback.

**Receive Incoming Calls**

The Mobile application receives incoming call data in Push Notification sent by your application backend. Refer section *Push Notification Endpoint* for details.




To Receive Push Notification from Push Notification Server, FirebaseMessaging must be initialized and subscribed for firebase message. refer Following PushNotificationService class which is provided as part of sdk.

```
// main.dart
main() async {
 await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
}
```

```
// PushNotificationService.dart
class PushNotificationService {

  static final PushNotificationService _instance = PushNotificationService._internal();

  FirebaseMessaging? _fcm;

  static PushNotificationService getInstance() {
    return _instance;
  }
  PushNotificationService._internal(){
    _fcm = FirebaseMessaging.instance;
  }

  Future initialize() async {
```

```
    FirebaseMessaging.onMessage.listen((RemoteMessage message) {
      ...
    }
    );
  }
  ...
}
```

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

```
FirebaseMessaging.onMessage.listen((RemoteMessage message) {
    exotelVoiceClient.relaySessionData(message.data);
});
```

in Above code,

1. you must inialize the `PushNotificationService` class
2. when push notification comes, data will be relayed to ExotelSDKClient using `relayFirebaseMessagingData()`
3. `ExotelSDKClient` will invoke `"relay-session-data"` Method.
4. on Method `"relay-session-data"`, Android/iOS native method handler will relay data to client SDK. which will be processes by client SDK and client SDK send callback `onIncomingCall()`.
5. Android/iOS native method handler will then invoke flutter Method `"on-incoming-call"`
6. on flutter Method `"on-incoming-call"`, `ExotelSDKClient` will send callback `onCallIncoming()`.

**Hangup**

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

```
exotelVoiceClient.hangup();
```

in Above code,

1. `ExotelSDKClient` will invoke `"hangup"` Method.
2. on Method `"hangup"`, Android/iOS native method handler will decline and terminate the call.

when call is successfully rejected,

- client SDK will send callback `onCallEnded()`
- Android/iOS native method handler will then invoke flutter Method `"on-call-ended"`.
- on flutter Method `"on-call-ended"`, `ExotelSDKClient` will check call status and then send `onCallEnded()` callback.

## 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 `ExotelSDKClient` object APIs.

```
// change audio route to speaker
exotelVoiceClient.enableSpeaker();
// change audio route to phone earpiece
exotelVoiceClient.disableSpeaker();
```

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

```
// enable the bluetooth
exotelVoiceClient.enableBluetooth();
// disable the bluetooth
exotelVoiceClient.disableBluetooth();
```

Local mic can be muted and unmuted during in-call using `ExotelSDKClient` object APIs.

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

## 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 `ExotelSDKClient` .

```
// 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.

## 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
exotelVoiceClient.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 |

# Platform Integration

## 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.

### 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

**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=743ddcf5a0552050bc37b6d0ff9613bn&CallFrom=sip:Alice&CallTo=sip:08040408080&CallStatus=ringing&Direction=i
ncoming&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&DialWhomNumber=&flow_id=249196&tenant_id=113828&From=sip:Alice&To=sip:08040408080&CurrentTime=2019-11-23+22:00:37
```

Response

JSON


```
{
    "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
}
```


**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/<accountsid>/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.

### Exotel API Endpoints

**Subscriber Management**

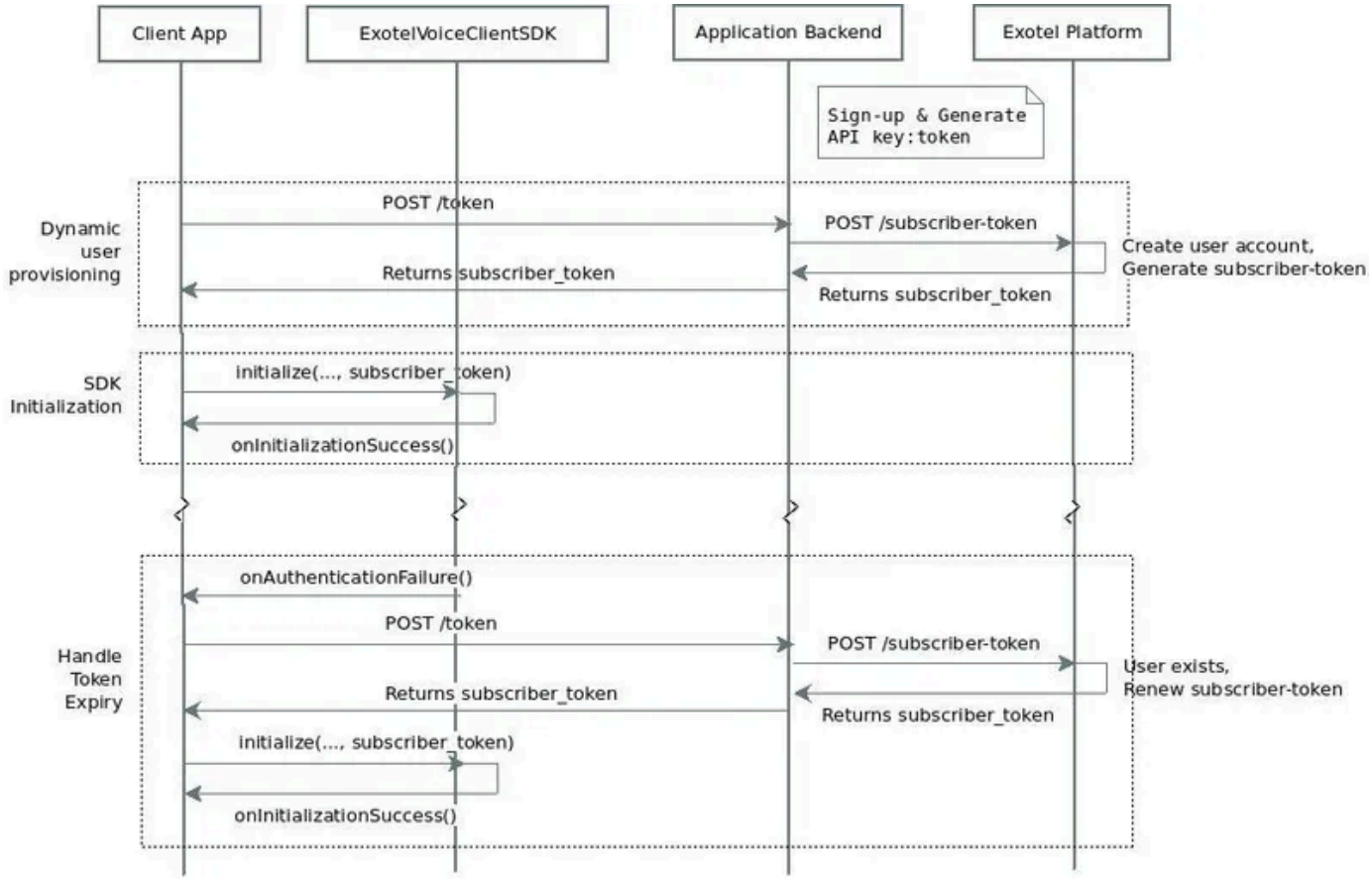
Customers can manage their subscribers in the Exotel platform using the APIs listed *"Subscriber-Management-API"* document.

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,

1. 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.
2. 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.

## 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

# 4.5.2. SDK Sample

| | |
|---|---|
| Sample SDK | **https://github.com/exotel/exotel-voip-sdk-flutter** |
| Latest Releases | **https://github.com/exotel/exotel-voip-sdk-flutter/releases** |

# 4.6. Subscriber Management API

## Introduction

This document defines the exotel platform APIs adding, updating, viewing and deleting the subscribers for VoIP Calling. Subscriber is the entity who uses the app with voip calling feature.

Subscriber provisioning in exotel can happen in two ways,

- Pre-provisioning

The customer pre-configures the client accounts even before the app is installed by the client.

- Dynamic provisioning

A client account is created after the app is installed by the user and signs-up with the application backend.

All the APIs listed in this document are privileged APIs that need API Key and API Token for authentication. API key:token are generated on sign-up with exotel platform. Please contact hello@exotel.in

***Base URL***

In all the APIs below, replace <base_url> with the following based on the stamp. Stamp info can be found in your Exotel dashboard under the API settings page.

- MUM Stamp - https://chaotix.mum1.exotel.in
- SGP Stamp - https://chaotix.apac-sg.exotel.in

## Subscriber Response Object

The subscriber object returned in the APIs is described below

| Parameter | Data Type | Description |
|---|---|---|
| subscriber_name | String | Subscriber name provided while creating the subscriber (Must be unique for a particular tenant)<br><br>Maximum supported subscriber name length is 16 bytes and can contain numeric characters. Whitespace characters are not allowed.<br>e.g. 123456, 9889899999 etc |
| status | Enum (active/ inactive) | Whether the subscriber is active or not |
| custom_field | String | Field to store custom metadata for the subscriber |
| date_created | DateTime | Time at which the subscriber was created |
| date_updated | DateTime | Time at which the subscriber was updated |

## Create Subscriber Token

Access to the exotel platform by ExotelVoiceClient is authenticated using a bearer token - *subsriber_token*. Clients must obtain this token from their application backend which in turn should fetch it from the exotel platform using the HTTPS endpoint listed here.

**Method:** `POST`

**Request URL:** `https://<base_url>/v2/accounts/<account-sid>/subscriber-token`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:**

| Parameter | Data Type | Description |
|---|---|---|
| platform | String | Parameter (Android / iOS) |
| device_id | String | Device id (AndroidID / iOS UDID) |
| subscriber_name | String | Unique subscriber identifier |

**Response Type:** `JSON`

**Response Code:**

`200 Success`

`400 Bad Request, Malformed Parameters`

`401 Unauthorized`

**Response Body Parameters**:

| Parameter | Data Type | Description |
|---|---|---|
| subscriber_token | String | Token to access exotel platform |
| subscriber_name | String | Subscriber name used for token generation |

**Note** - The Time-to-Live (TTL) for token expiry is as follows:

Refresh token expiry: 24 hours

Access token expiry: 30 minutes

Sample Request

JS

```
https://<base_url>/v2/accounts/exotel13m2/subscriber-token
{
"platform": "android",
"device_id": "2fc4b5912826ad1",
"subscriber_name": “archit”
}
```

Sample Response

JSON


```
{
"request_id": "2f97c5e98e1d4c188c483cbfe8092869",
"method": "POST",
"http_code": 200,
“response”:
{
"code": 200,
"error_data": null,
"status": "success",
“data”:
{
    "subscriber_name": “archit”,
"subscriber_token":
{
"refresh_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3
"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJleG90ZWwiLCJzdWIiOiJBcmNoaXQiLCJpYXQiOjE1NzY2NDc5OTksImV4cCI6MTU3N
}
},
}
}
```


## Create Subscribers

API is used for both single or bulk subscriber account creation in exotel platform.

**Method:** `POST`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters**:

| Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| subscriber_name | String | Mandatory | Unique Username for the Subscriber |
| custom_field | String | Optional | Field to store custom metadata for the Subscriber |

Please note that POST is a bulk operation and the request is actually an array of the above.

The maximum number of subscribers which can be created in a single request is 10.

**Response Type:** `JSON`

**Response Body Parameters:** Returns multipart response for request. Refer Subscriber Response Object

Sample Request

JS

```
https:///<base_url>/v2/accounts/<accountsid>/subscribers
{
  "subscribers": [
    {
      "subscriber_name": "bob",
      "custom_field": "support"
    },
    {
      "subscriber_name": "nidhi"
    },
    {
      "subscriber_name": "archit"
    }
  ]
}
```

Sample Response

JSON


```
{
"request_id": "04386e350256448dae83c6db0f749a21",
"method": "POST",
"http_code": 207,
"metadata": {
"failed": 0,
"success": 3
},
"response": [
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:17+05:30",
"date_updated": "2019-11-19T23:14:17+05:30",
"subscriber_name": "bob",
"status": "active",
"custom_field": "support”
}
},
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:17+05:30",
"date_updated": "2019-11-19T23:14:17+05:30",
"subscriber_name": "nidhi",
"status": "active",
"custom_field": null
}
},
{
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-19T23:14:18+05:30",
"subscriber_name": "archit",
"status": "active",
"custom_field": null
}
}
],
}
```


## Update a Subscriber

API is used to update the subscriber account parameters after it has been created. Subscriber can be marked inactive for temporary suspension.

**Method:** `PUT`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber-name>`

**Authorization Type:** Basic authentication using API key:token

**Request Body Parameters**:

| Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| custom_field | String | Optional | Field to store custom metadata for the subscriber |
| status | Enum | Mandatory | Indicates whether the subscriber is active or not<br>enum {"active", "inactive"} |

**Response Type:** JSON

**Response Body Parameters:** Refer Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/archit
{
  "status": "inactive",
}
```

Sample Response

JSON

```
{
"request_id": "8019271c8d98422db509c2801e63435f",
"method": "PUT",
"http_code": 200,
"response": {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-20T11:13:38+05:30",
"subscriber_name": "archit",
"status": "inactive",
"custom_field": null
}
}
}
```

## Get Subscriber

API returns subscriber details.

**Method:** `GET`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber_name>`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:** None

**Response Type:** `JSON`

**Response Body Parameters:** On success returns Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/archit
```

Sample Response

JSON


```
{
"request_id": "287c227674af40b38531a1c247262deb",
"method": "GET",
"http_code": 200,
"response": {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-11-19T23:14:18+05:30",
"date_updated": "2019-11-20T11:13:38+05:30",
"subscriber_name": "archit",
"status": "inactive",
"custom_field": null
}
}
}
```


## List Subscribers

API returns a paginated list of subscribers.

**Method:** `GET`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers>?status=abc&page_size=xyz&offset=xyz`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:**

| Query Parameter | Data Type | Requirement | Description |
|---|---|---|---|
| status | Enum | Optional | Status for the subscriber |
| page_size | uint64 | Optional | Page size for response. Default and maximum value is 100 |
| offset | uint64 | Optional | Offset from which to get the records. Defaults to 0 |

**Response Type:** `JSON`

**Response Body Parameters:** On success returns paginated list of Subscriber Response Object

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers?page=2&offset=3
```

Sample Response

JSON


```
{
"request_id": "b5f0c6780ec443d08947ba2015a06a04",
"method": "GET",
"http_code": 200,
"metadata": {
"prev_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=6&page_size=3",
"next_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=12&page_size=3",
"first_page_uri": "/v2/accounts/<accountsid>/subscribers?offset=0&page_size=3"
},
"response": [
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T17:12:00+05:30",
"date_updated": "2019-12-30T17:12:00+05:30",
"subscriber_name": "shyam",
"status": "active",
"custom_field": null
}
},
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T17:09:41+05:30",
"date_updated": "2019-12-30T17:09:41+05:30",
"subscriber_name": "kiran",
"status": "active",
"custom_field": "test"
}
},
  {
"code": 200,
"error_data": null,
"status": "success",
"data": {
"date_created": "2019-12-30T13:35:07+05:30",
"date_updated": "2019-12-30T13:35:07+05:30",
"subscriber_name": "mandeep",
"status": "active",
"custom_field": null
}
}
],
}
```


## Delete Subscriber

API deletes a subscriber permanently.

**Method:** `DELETE`

**Request Url:** `https://<base_url>/v2/accounts/<account-sid>/subscribers/<subscriber-name>`

**Authorization Type:** `Basic authentication using API key:token`

**Request Body Parameters:** None

**Response Type:** `JSON`

**Response Body Parameters:** On success, HTTP response 204 is returned with no response body. In case of failure 404 is returned

Sample Request

JS

```
https://<base_url>/v2/accounts/<accountsid>/subscribers/shyam
```

Sample Response

(a) Success: `No body`

(b) Failure:

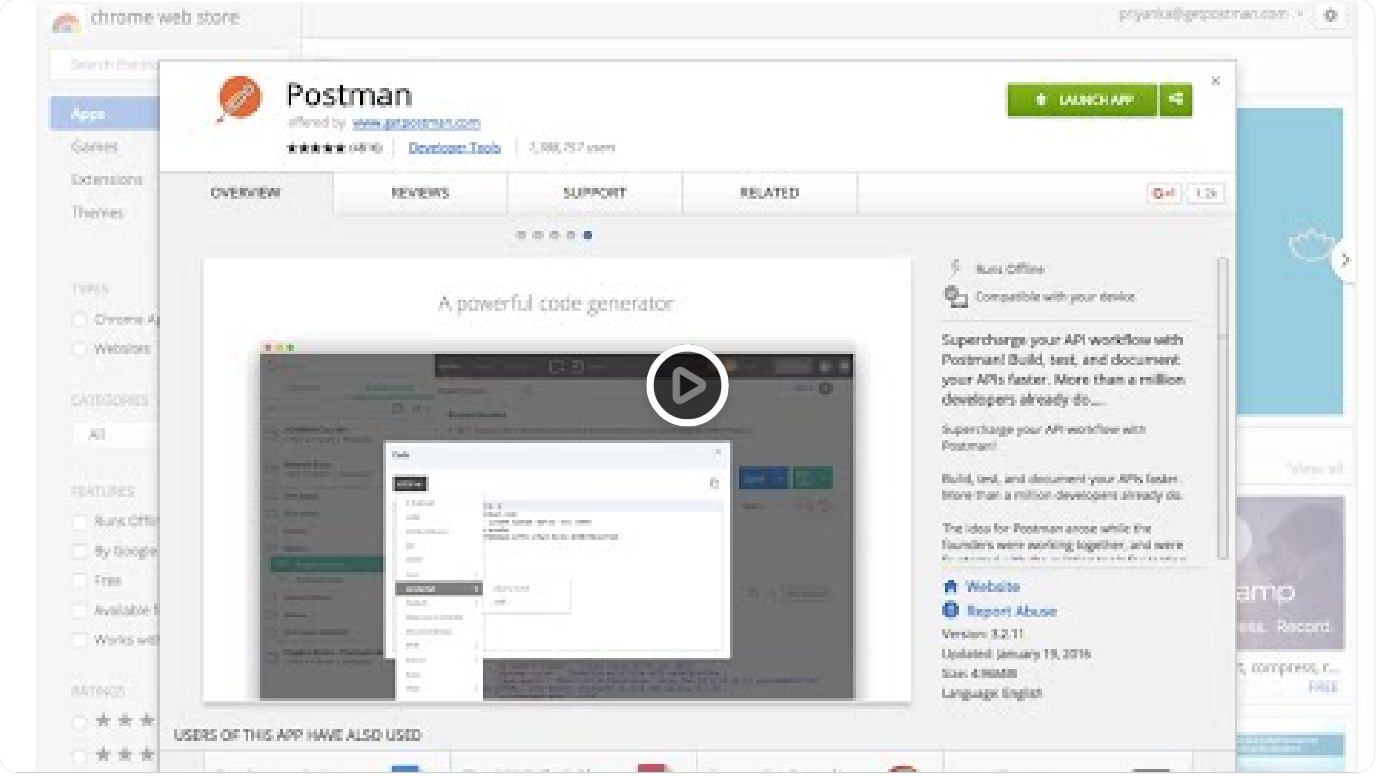
JSON

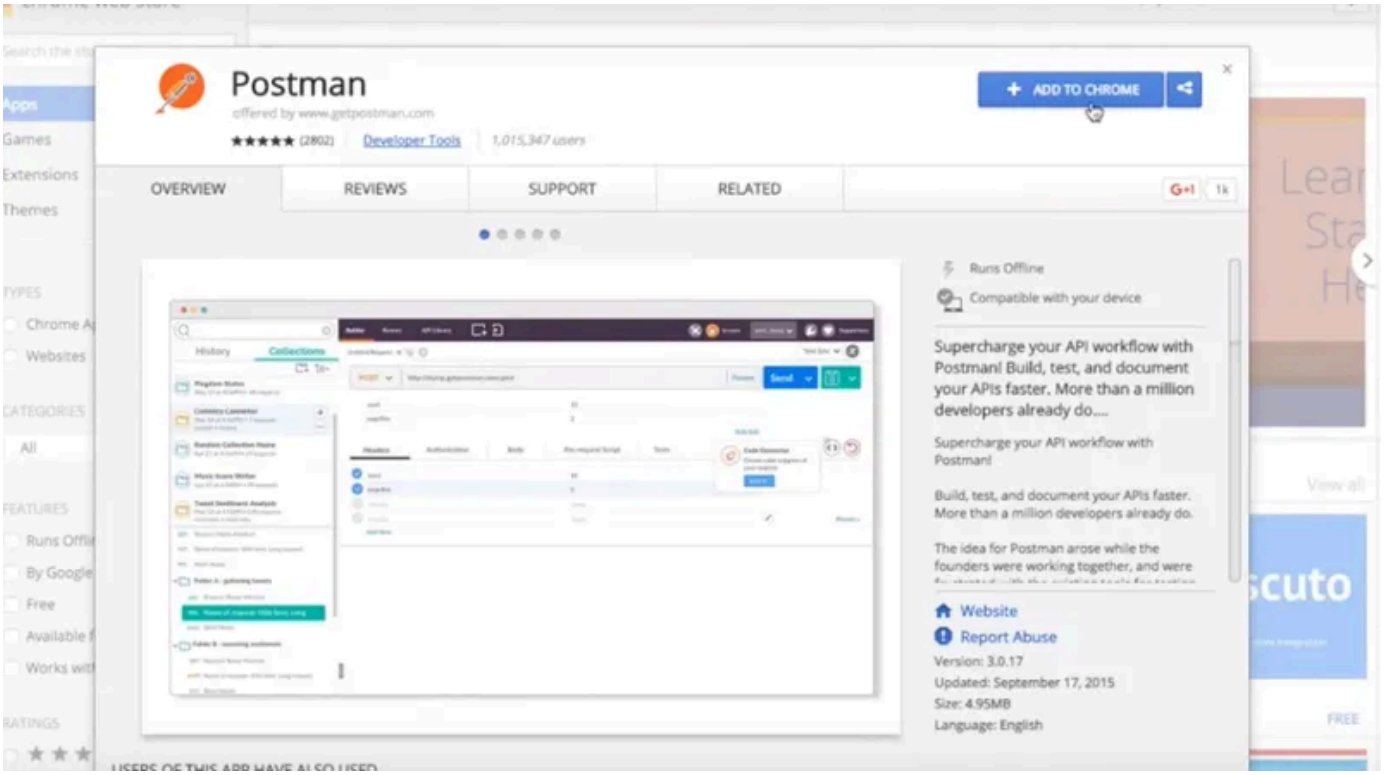
```
{
    "request_id": "af0087d5331f479599d2c987efe9f664",
    "method": "DELETE",
    "http_code": 404,
    "response": {
        "code": 404,
        "error_data": {
            "code": 1000,
            "description": "Subscriber not found",
            "message": "Not Found"
        },
        "status": "failure",
        "data": null
    }
}
```

# 5. Additional API

## 5.1. Make an API Outgoing call using Postman

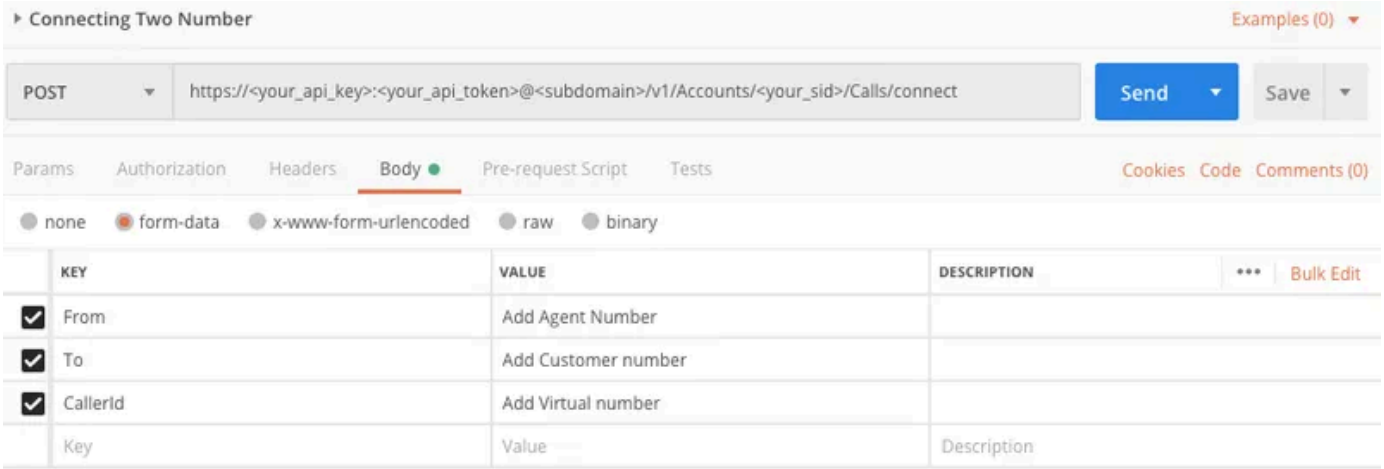
You can use **Postman - a free Google Chrome Extension** - to make an outgoing call using our **Call APIs**, and the steps to do so are as follows:






If the plugin is not working, you can use this function directly by downloading the free Postman App.

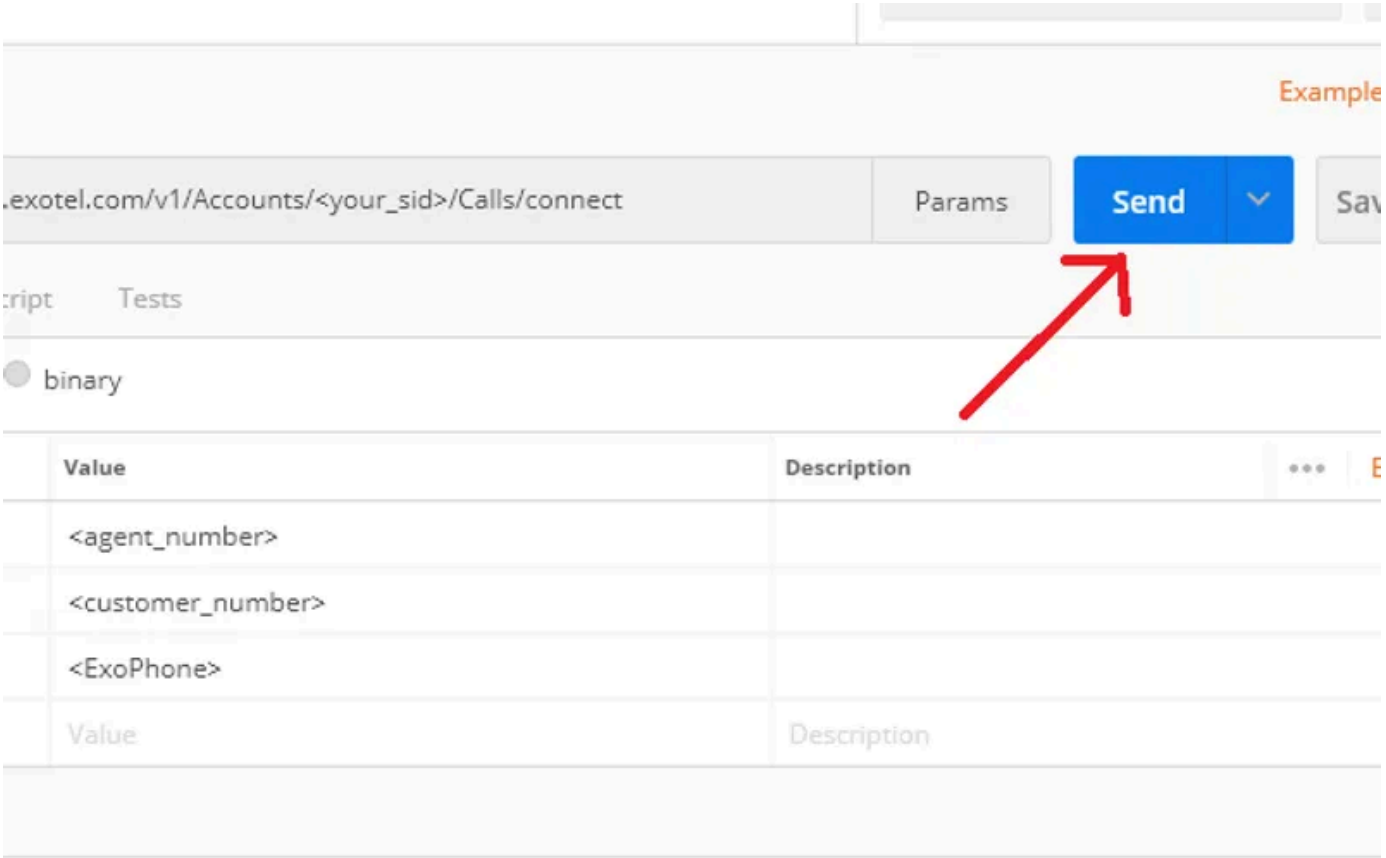
**Step 2** - Sign up and begin importing this **collection configuration** after launching the app (by clicking on **https://www.getpostman.com/collections/da36b086217bca1fb732**, the collection's settings will be automatically imported to your Postman app/account).




**Step 3** - Replace all the following variables:

| | |
|---|---|
| <your_api_key>, <your_api_token>, <your_sid> | **How to find my Exotel Key, Exotel Token and Exotel SID?** |
| <agent_number> | Enter the 'From' or the caller's number, who would answer the call first |
| <customer_number> | Enter the 'To' or the callee's number, to whom the caller/agent is trying to connect |
| <ExoPhone> | Enter the virtual number / DID through which you wish to route the call |

**Step 4** - Click 'Send'




## Watch a short video

You are not permitted to perform this action

# 5.2. StatusCallback reliability: expected latency, retry logic, and handling missed callbacks

## Overview

When a call ends, Exotel sends an HTTP StatusCallback to your configured StatusCallback URL with the final call status and related details, including the recording URL when available.

For most calls, this callback arrives shortly after the call completes. For a small percentage of calls (typically under 1%), the callback may be missing, delayed, or incomplete. This article covers delivery behavior, retries, expected latency, and the recommended fallback.

### Expected latency

StatusCallback fires when a call reaches a final state (for example, completed, failed, busy, no-answer, or canceled).

Under normal conditions, expect delivery within a few seconds to under a minute after the call ends. In practice, delivery is typically well within 45 seconds after the callback is queued.

Note: the recording URL may still be unavailable on the first callback even when the status is final. It can arrive later, once the recording is ready.

### Delivery retries to your endpoint

If Exotel can't deliver the StatusCallback to your endpoint (for example, due to a timeout, a temporary 5xx, or an HTTP 429), it retries delivery:

| Detail | Value |
|---|---|
| Maximum delivery attempts | Up to 4 |
| Timeout per attempt | 6 seconds |
| Retry spacing (on timeout) | ~30 seconds → 5 minutes → 10 minutes |
| Other temporary failures (5xx / 429) | Retried with a short delay (typically ~30 seconds) |
| Permanent client errors (most 4xx, except 429) | Not retried |

Your endpoint should respond with a 2xx (or a 3xx redirect) within 6 seconds for the delivery to count as successful.

### When StatusCallback may be missing, late, or incomplete

In rare cases, complete call details aren't available in time, due to delays in the telecom/network path or in call finalization. When this happens, you may see one or more of the following:

- StatusCallback isn't delivered.
- StatusCallback arrives with incomplete fields (for example, status or leg details are null/empty).
- The recording URL is missing or arrives later than usual.

This is expected edge-case behavior for a small fraction of calls, and doesn't usually indicate a misconfiguration on your end.

**Important:** The retries described above only cover delivery failures to your URL (your server being down or slow). They don't cover the rarer case where Exotel never queues a complete final callback because call details weren't available in time. For that case, use Get Call

Details.

## Recommended fallback: Get Call Details API

Don't rely on StatusCallback alone for 100% coverage. Use Get Call Details as the source of truth whenever a callback is missing or incomplete:

1. If you don't receive a StatusCallback for a CallSid, call Get Call Details for that CallSid.
2. If the status or recording URL is still empty or incomplete, retry with backoff – for example, start 1–2 minutes after the call, then retry every few minutes.
3. As a practical guideline, check again around 15 minutes after the call was placed or completed. By then, most calls have a final status available.
4. Stop retrying once you have a final status (completed, failed, busy, no-answer, canceled, etc.), or once you hit a maximum retry window you define (for example, 30–60 minutes).

## Recommended integration pattern

- Use StatusCallback for near-real-time updates.
- Make sure your StatusCallback endpoint responds quickly (within 6 seconds) with a 2xx.
- Use Get Call Details (with retries) to reconcile missed or incomplete callbacks.
- Optionally, run a periodic reconciliation job for CallSids you initiated but never fully resolved.

## Summary

| Topic | Guidance |
|---|---|
| Happy-path latency | Usually seconds to under a minute after call end |
| Delivery retries | Up to 4 attempts, with backoff |
| Per-attempt timeout | 6 seconds |
| Missed/incomplete edge cases (<1%) | Use Get Call Details with retries (~15-minute practical window) |
| Recording URL | May lag behind final status – poll Get Call Details if needed |

# 5.3. How to perform Listen Whisper Barge using LWB API

This document outlines APIs for performing LWB, to help supervisors perform monitor action on an ongoing call. In order to perform the action, this document describes the below two endpoints which a developer shall use to perform the required action from a supervisor.

1. Get the active legs of the ongoing call: Typically an ongoing call shall have two legs, 1 from the customer and 1 from the agent. This API shall return the leg details in order for the developer to identify which is the agent leg to perform the monitor action.
2. Create the monitor leg: Once the leg is identified, one needs to create another leg on the call to perform the required action (Listen / Whisper / Barge).

*NOTE: In Order to use these APIs, one needs to provide the callSID of the ongoing call. This is a dependency on the API. To enable this API in your account, please reach out to hello@exotel.in*

**1. Get the active legs of the ongoing call**

An HTTP GET request to has to be made

https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/<your_sid>/Calls/<CallSid>/ActiveLegs

- Replace <your_api_key> and <your_api_token> with the API key and token of your account.
- Replace <your_sid> with your “Account sid”
- Replace <subdomain> with the region of your account
  - <subdomain> of Singapore cluster is api.exotel.com
  - <subdomain> of Mumbai cluster is api.in.exotel.com
- Replace <CallSid> with the "Call Sid of your ongoing call"

<your_api_key> , <your_api_token> and <your_sid> are available in the API settings page of your Exotel Dashboard

**Sample request :-**

curl https://<your_api_key>:<your_api_token>

@api.in.exotel.com/v1/Accounts/<your_sid>/Calls/<CallSid>/ActiveLegs.json

**HTTP Response:**

On success,

- the HTTP response status code will be 200
- the HTTP body will contain an XML similar to the one below, an array of legs

JSON


```
{
"Legs":[
{
"Sid":"XXXXXXXXXX",
"CallSid":"XXXXXXXX",
"AccountSid":"XXXXXX",
"PhoneNumber":"xxxx",
"Status":"in-progress",
"Origin":"outbound",
"LastAction":"",
"DateCreated":"2020-12-22 17:31:06",
"DateUpdated":"2020-12-22 17:31:12",
"Uri":"/v1/Accounts/XXXXX/Calls/XXXXXXX/ActiveLegs"
},
{
"Sid":"XXXXX",
"CallSid":"XXXXX",
"AccountSid":"XXXXX",
"PhoneNumber":"xxxx",
"Status":"in-progress",
"Origin":"outbound-dial",
"LastAction":"",
"DateCreated":"2020-12-22 17:31:13",
"DateUpdated":"2020-12-22 17:31:13",
"Uri":"/v1/Accounts/XXXXX/Calls/XXXXX/ActiveLegs"
}
```


Here is the **Sid** if the Leg identifier is to be used in creating the monitor action on a leg.

**On Failure,**

<?xml version="1.0" encoding="UTF-8"?>

<TwilioResponse>

<RestException>

<Code>404</Code>

<Message>Given callSID is not in-progress</Message>

</RestException>

</TwilioResponse>sh-3.2#

**2. Create the monitor leg**

This is the supervisor leg which will get created on successful execution of the below API. On 200 OK, a call attempt shall be made from the Exotel platform to the supervisor and on pick up of the call, the supervisor shall join the ongoing call in "Listen" "Whisper" or a "Barge" mode.

**An HTTP POST request to https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/<your_sid>/Calls/<CallSid>/Legs**

**The following are the POST parameters:**

| Field | Mandatory/ Optional | DataType | Values |
|---|---|---|---|
| Action | Mandatory | string | “listen”<br>“whisper”<br>“barge” |
| PhoneNumber | Mandatory | string | A new leg will be created for this phone number.<br>If mobile number, prefix the 10 digits with a 0; Ex:✆✆ 09052161119.<br>If landline number, prefix it with STD code; Ex: 08030752400 |
| TargetLegSid | Optional | string | Logical Leg ID of the leg on which the action is to be performed.<br>Mandatory if Action = whisper |
| StatusCallback | Optional | string | When the call completes, an HTTP POST will be made to the URL mentioned below with the parameters |
| StausCallbackEvents | Optional | Array | An array of events for which StatusCallback has to be sent. Currently, only "terminal" is supported |
| CustomField | Optional | string | Application-specific value which will passed in the statuscallback |
| StatusCallbackContentType | Optional | string | Allowed values<br>multipart/form-data (default)<br>application/json |

HTTP Response:

On success,

- the HTTP response status code will be 200
- the HTTP body will contain an XML similar to the one below. The "Sid" is the unique identifier of the Leg and it will be useful for any future action on this leg

<?xml version="1.0" encoding="UTF-8"?>

<TwilioResponse>

<Leg>

<Sid>XXXXXXXXXXXXXXXXX</Sid>

<CallSid>XXXXXXXXXXXXXXX</CallSid>

<AccountSid>XXXXXXXX</AccountSid>

<PhoneNumber>XXXXXXXXX</PhoneNumber>

<Status>in-progress</Status>

<Origin>monitor</Origin>

<LastAction>barge</LastAction>

<DateCreated>2020-12-22 17:57:40</DateCreated>

<DateUpdated>2020-12-2217:57:40</DateUpdated>

<StartTime>2020-12-2217:57:40</StartTime>

<Uri>/v1/Accounts/XXXXXX/Calls/XXXXXXX/Legs</Uri>

</Leg>

</TwilioResponse>

**On failure,**

- the HTTP response status code will be non-200.
- the HTTP body will contain an XML (such as the one below) with details of why the request failed.

<?xml version="1.0" encoding="UTF-8"?>

<TwilioResponse>

<RestException>

<Status>400</Status>

<Message>Invalid Parameter: PhoneNumber is missing or invalid</Message>

</RestException>

</TwilioResponse>

**Sample HTTP request**

curl -H "Content-Type: application/json" --request POST -d '{"Action":"barge","TargetLegSid":"","PhoneNumber":"xxxx","StatusCallback":"","CustomField":"LegsCustom","StatusCallbackContentType":"application/json","StatusCallbackEvents[0]":"terminal"}' https://<Your_account_key>:<Your_account_token>@api.in.exotel.com/v1/Accounts/exotel30m/Calls/<CallSID>/Legs

**StatusCallback**

Exotel will perform an asynchronous HTTP request to the StatusCallback URL you have specified in your request (if any) once the call completes. List of parameters that will be sent as part of StatusCallback:

Sample StatusCallback payload for 'terminal' event:

"LegSid": "XXXXXXX"

"CallSid": "XXXXXX"

"AccountSid": "XXXXXX"

"DateCreated": "2021-02-08 19:33:59"

"DateUpdated": "2021-02-08 19:34:32"

"StartTime": "2021-02-08 19:33:59"

"EndTime": "2021-02-08 19:34:32"

"PhoneNumber": "XXXXXXXX"

"Status": "completed"

"OnCallDuration": "52"

"Origin": "monitor"

"LastAction": "whisper"

"CustomField": "LegsCustom"

"EventType": "terminal"

**3. Upgrade the monitor Leg**

Once an action like Listen or Whisper, one can decide to perform the next action like Barge on the same leg created. To support this use case, API supports an upgrade capability where only below fields shown in the table below are accepted

***Please note that the monitor call leg should not be disconnected while performing this PUT request. Else you will receive an error that the Leg sid does not exist.***

An HTTP PUT request to has to be made

https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/<your_sid>/Calls/<CallSid>/Legs/<Monitor_leg_SID>

- Replace **<your_api_key>** and **<your_api_token>** with the API key and token of your account.
- Replace **<your_sid>** with your “Account sid”
- Replace **<subdomain>** with the region of your account
  - <subdomain> of Singapore cluster is api.exotel.com
  - <subdomain> of Mumbai cluster is api.in.exotel.com
- Replace **<CallSid>** with your “Call sid of the ongoing call”
- Replace **<Monitor_leg_SID>** with your “Leg sid of the Monitor API”

**<your_api_key>** , **<your_api_token>** and **<your_sid>** are available in the API settings page of your Exotel Dashboard

**The following are the PUT parameters:**

| Field | DataType | Mandatory /Optional | Description |
|---|---|---|---|
| Action | string | Mandatory | A new state for the call. Supported values whisper/ barge.<br><br>The only transition allowed are:<br>listen ->whisper<br>whisper-> barge<br>listen -> barge |
| TargetLegSid | string | Mandatory | Logical Leg ID of the leg on which the action is to be performed.<br>Mandatory if Action = whisper<br>Optional if Action = barge |

**Sample request**

curl -H "Content-Type: application/json" --request PUT -d '{"Action":"barge"}' https://<Your_API_Key>:<Your_API_Token>@api.in.exotel.com/v1/Accounts/<your_sid>/Calls/<Youd_Call_Sid>/Legs/<Monitor_leg_SID>

**HTTP Response:**

On success :-

- the HTTP response status code will be 200
- the HTTP body will contain an XML similar to the one below.

<?xml version="1.0" encoding="UTF-8"?>

<TwilioResponse>

<Leg>

<Sid>XXXXXX</Sid>

<CallSid>XXXXXX</CallSid>

<AccountSid>XXXXX</AccountSid>

<PhoneNumber>XXXXXX</PhoneNumber>

<Status>in-progress</Status>

<Origin>monitor</Origin>

<LastAction>barge</LastAction>

<DateCreated>2020-12-24 10:55:54</DateCreated>

<DateUpdated>2020-12-24 10:56:53</DateUpdated>

<StartTime>2021-02-08 19:33:59</StartTime>

<Uri>/v1/Accounts/XXXXXX/Calls/XXXX/Legs/XX</Uri></Leg></TwilioResponse>

On Failure:

- the HTTP response status code will be non-200.
- the HTTP body will contain an XML (such as the one below) with details of why the request failed.

<?xml version="1.0" encoding="UTF-8"?>

<TwilioResponse>

<RestException>

<Code>404</Code>

<Message>Given legSID does not exist. Please check the legSID passed</Message>

</RestException>

</TwilioResponse>

**Get all legs on monitor leg creation success**

After the monitor leg is created, the developer can hit the Get the active legs endpoint to check the current status of all the ongoing legs.

**Sample request:**

curl https://<your_api_key>:<your_api_token>

@api.in.exotel.com/v1/Accounts/<your_sid>/Calls/<>Call_SID/ActiveLegs.json

**Sample response:**

{

"Legs":[

{

"Sid":"XXXX",

"CallSid":"XXXX",

"AccountSid":"XXXX",

"PhoneNumber":"XXXX",

"Status":"in-progress",

"Origin":"outbound",

"LastAction":"",

"DateCreated":"2020-12-22 17:56:55",

"DateUpdated":"2020-12-22 17:57:02",

"Uri":"/v1/Accounts/XXXX/Calls/XXXX/ActiveLegs"

},

{

"Sid":"XXXX",

"CallSid":"XXXX",

"AccountSid":"XXXX",

"PhoneNumber":"XXXX",

"Status":"in-progress",

"Origin":"outbound-dial",

"LastAction":"",

"DateCreated":"2020-12-22 17:57:03",

"DateUpdated":"2020-12-22 17:57:03",

"Uri":"/v1/Accounts/XXXX/Calls/XXXX/ActiveLegs"

},

{

"Sid":"XXXX",

"CallSid":"XXXX",

"AccountSid":"XXXX",

"EndUserNumber":"XXXX",

"Status":"in-progress",

"Origin":"monitor",

"LastAction":"barge",

"DateCreated":"2020-12-22 17:57:40",

"DateUpdated":"2020-12-22 17:57:40",

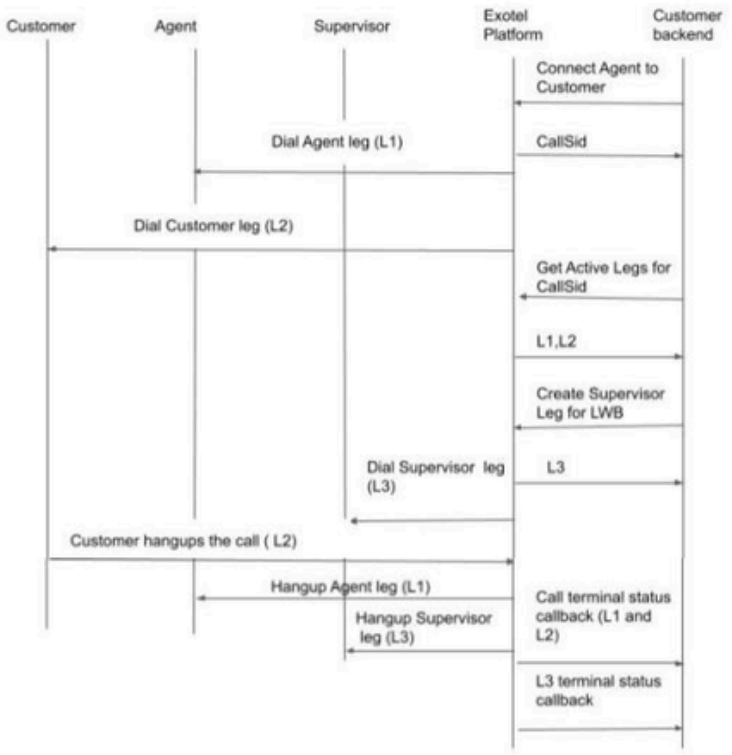
"Uri":"/v1/Accounts/XXXX/Calls/XXXX/ActiveLegs"

}

]

}

**A typical call flow for LWB API in outbound API connecting agent to customer use case**




**Call Disconnect API <Hangup>**

**High-Level Approach**

1. Hangup action will be enabled in the ExoML API.
2. Customer to query legs for the callSid and perform hangup action on leg

**API Contract:**

An HTTP PUT request to **https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/<your_sid>/Calls/<CallSid>/Legs/<LegSid>**

**The following are the PUT parameters:**

| Field | Mandatory/ Optional | DataType | Values |
|---|---|---|---|
| Action | Mandatory | string | “hangup” |

**Sample Request**

```
curl --location --request PUT 'http://<API_KEY>:<API_Token>@api.in.exotel.com/v1/Accounts/<Account_SID>/Calls/<Call_SID>/Legs/<Leg_SDI>' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'Action=hangup'
```

**Sample Response: JSON Response (Default response type is XML)**

```
{
"legs": {
"Sid": "80de40c29dXXX83b485f8d4168c",
"CallSid": "9aeaea05XXXX268d3c872168c",
"AccountSid": "exXXXt",
"PhoneNumber": "0720XX45365",
"Status": "in-progress",
"Origin": "outbound",
"LastAction": "hangup",
"DateCreated": "2022-08-12 12:14:44",
"DateUpdated": "2022-08-12 12:14:58",
"StartTime": "2022-08-12 12:14:56",
"ExoPhone": "080XXX92531",
"BackupExoPhone": "0804XXX2531",
"Uri": "/v1/Accounts/exXXXIt/Calls/9aeaea053872168c/Legs/80de40c29d3b485f8d4168c"
}
}
```

**Sample Response : XML Response**

```
<?xml version="1.0" encoding="UTF-8"?>
<TwilioResponse>
 <Legs>
 <Sid>80de40c29d9d8d4168c</Sid>
 <CallSid>9aeaea053de5268d3c872168c</CallSid>
 <AccountSid>abjcw</AccountSid>
 <PhoneNumber>072XX45365</PhoneNumber>
 <Status>in-progress</Status>
 <Origin>outbound</Origin>
 <LastAction>hangup</LastAction>
 <DateCreated>2022-08-12 12:14:44</DateCreated>
 <DateUpdated>2022-08-12 12:14:58</DateUpdated>
 <StartTime>2022-08-12 12:14:56</StartTime>
 <ExoPhone>0804XX2531</ExoPhone>
 <BackupExoPhone>0804XX92531</BackupExoPhone>
 <Uri>/v1/Accounts/sdcds/Calls/9aeaea053de5268d3c872168c/Legs/80de40c29d9d645bd185f8d4168c</Uri>
 </Legs>
</TwilioResponse>
```

**Sample Failure Response: XML**

```
<?xml version="1.0" encoding="UTF-8"?>
<TwilioResponse>
 <RestException>
 <Code>404</Code>
 <Message>Not Found</Message>
 </RestException>
</TwilioResponse>
```

**Sample Failure Response: JSON**

```
{
"RestException": {
"Code": 404,
"Message": "Not Found"
}
}
```

**Billing**

The supervisor leg will be charged as per the bill plan for calls applied to your account.

## Watch a short video

# 5.4. Enabling Applet-Level Call Recording

Enabling Applet-Level Call Recording

## Overview

This guide demonstrates how enterprises can dynamically control call recording within their Exotel applet flow using the **Passthru Applet** (GET request) and **Recording API** (POST request). This setup enables selective, event-driven recording control – for example, starting the recording before a greeting or IVR prompt and stopping it before agent transfer.

The implementation combines the**Passthru Applet** and**Call Recordings** to allow real-time backend-triggered recording.

## How It Works

1. **Use Passthru Applet (GET):** The Passthru applet sends call information such as CallSID, From, and To parameters to your backend via a GET request.
2. **Backend Triggers Recording API (POST):** Your backend receives the GET request, extracts the CallSID, and immediately makes a POST call to the **Recording API** to start recording.
3. **Continue Call Flow:** The call proceeds through the configured applets (Greeting, IVR, Gather, etc.) while recording remains active.
4. **End-of-Flow Stop Recording:** At the final stage, another Passthru applet sends a GET request to your backend endpoint (e.g., /stop_recording), which triggers a POST API call to stop recording.

This ensures Exotel only records the specific portion of the call you need, such as the main interaction, and not hold or transfer periods.

## Step-by-Step Implementation

## 1. Configure the Passthru Applet (Start Recording)

- Add a **Passthru Applet** at the point in the flow where you want to begin recording.
- Set the Passthru applet's GET **URL** to a backend endpoint like https://yourdomain.com/start_recording.
- When executed, Exotel performs a **GET** request to this endpoint with call details appended as query parameters:

```
https://yourdomain.com/start_recording?
CallSid=abcd1234efgh5678&From=%2B919900112233&To=08088919888&CallType=outgoing
```

Your backend should capture the CallSid from this GET request quer params

## 2. Backend: Invoke Start Recording API (POST)

Your backend now calls Exotel's Recording API using the captured CallSid.

`POST`

```
https://<your_api_key>:<your_api_token><subdomain>/v1/Accounts/<your_sid>/Calls/<CallSID>/recording.json
```

**Example:**

Curl

```
curl --location 'https://<base_url>/v1/Accounts/<AccountSID>/Calls/<CallSID>/recording.json' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'Action=START' \
--data-urlencode 'RecordingChannels=dual' \
--data-urlencode 'RecordingFormat=mp3-hq' \
--data-urlencode 'StatusCallback=<status_callback_url>' \
--data-urlencode 'Leg1Recording=True'
```

Replace `<your_api_key>` and `<your_api_token>` with the API key and token created by you.

- Replace `<your_sid>` with your "Account sid"
- Replace `<subdomain>` with the region of your account
    1. `<subdomain>` of Singapore cluster is @api.exotel.com
    2. `<subdomain>` of Mumbai cluster is @api.in.exotel.com

`<your_api_key>` , `<your_api_token>` and `<your_sid>` are available in the API settings page of your Exotel Dashboard

**Parameter Reference:**

| Parameter | Type | Description |
|---|---|---|
| **Action** | String | START or STOP to control recording. |
| **RecordingChannels** | String | Optional – Audio channels. • single *(default)* – Both legs mixed. • dual – Caller/callee recorded separately. |
| **RecordingFormat** | String | Optional – Audio quality. • mp3 *(default)* – Standard bitrate. • mp3-hq – High bitrate (32 kbps/channel). Requires enablement via hello@exotel.com. |
| **StatusCallback** | URL | Callback endpoint for recording status. |
| **Leg1Recording** | Boolean | True to record only the first call leg. |

## 3. Continue with Business Applets

After triggering the start recording API, your flow continues normally with applets such as:

- **Greeting Applet:** Playback welcome message.
- **IVR Applet:** Capture user selections.
- **Gather Applet:** Collect input.
- **Connect Applet:** Transfer to agent.

**Example Flow:**

```
Passthru (GET → start_recording) → Backend (POST → Start Recording API) → Greeting → IVR → Gather → Passthru (GET → stop_recording)
```

## 4. Configure Passthru Applet (Stop Recording)

At the end of the flow:

- Add another **Passthru Applet** configured to call a URL like `https://yourdomain.com/stop_recording.`
- This GET request from Exotel again includes the CallSid.
- Your backend receives this GET, extracts the CallSid, and invokes:

Curl

```
curl --location 'https://<your_api_key>:<your_api_token><subdomain>/v1/Accounts/<your_sid>/Calls/<CallSID>/recording.json' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'Action=STOP'
```

This cleanly terminates recording at the end of the call.

## Recording Access

- **Dashboard:** Under each call log.
- **Call Detail APIs: **https://developer.exotel.com/api/make-a-call-api#call-details

## Compliance Notes

- The Passthru applet always sends a **GET request** to your backend; the enterprise must use POST to trigger or stop recording.
- Obtain customer consent per regional compliance rules.
- *mp3-hq* is an optional, on-demand feature.
- Recordings are stored securely in Exotel's environment.

## Example Use Case

**Scenario:** A customer support survey where only the IVR interaction is recorded.

1. Passthru applet sends GET to backend → backend triggers **Start Recording (POST)**.
2. IVR runs and records customer input.
3. Final Passthru sends GET → backend triggers **Stop Recording (POST)** before agent connection.

This ensures only the main customer interaction is captured.

## References

**Passthru Applet**

**Call Recordings**

**Secure Call Recording**

# 5.5. Leg details API to get all participants details of a Call

This document outlines an API to query all the participants in the completed calls. The use case where this API will be useful

1. Calls where more than 2 participants were there in a call. For Ex:- Dialer, Cold transfer, Connect after connect
2. Listen, Whisper or Barge use case. More details of LWB API is here

**NOTE:- In Order to use this API, one needs to provide the callSID of the completed call. This is a dependency for the API**

**NOTE:- To enable this API in your account, please reach out to hello@exotel.com**

To get the details of a call , you will need to make a HTTP GET request to

https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/~exotel_sid~/Calls/<CallSid>/Legs

- <CallSid> is the Sid of the call.
- Replace <your_api_key> and <your_api_token> with the API key and token created by you.
- Replace <your_sid> with your “Account sid”
- Replace <subdomain> with the region of your account
    - <subdomain> of Singapore cluster is api.exotel.com
    - <subdomain> of Mumbai cluster is api.in.exotel.com

<your_api_key> , <your_api_token> and <your_sid> are available in the API settings page of your Exotel Dashboard

In the case of an outbound call originating via the Call APIs (here or here), the "Sid" parameter is returned in the XML response to your request.

In the case of inbound calls, you can get the CallSid by using the Passthru applet.

**HTTP Response:**

**On success**, this API will return an HTTP status code of 200 and the response body will contain an XML/JSON (like below) that contains the details of the call.

**On failure**, the HTTP status code will be a non-200 code which indicates the reason (as discussed here ).

**Response Body on Success:**

JSON

```
{
"MetaData":{
"Total":3,
"Count":3
},
"legs":[
{
"Sid":"XXXXXXXXXXXXXX",
"CallSid":"XXXXXXXXXXXXXX",
"AccountSid":"exotelt",
"PhoneNumber":"XXXXXXXXXXXXX",
"Status":"completed",
"Origin":"outbound",
"LastAction":"",
"DateCreated":"2021-06-24 11:12:39",
"DateUpdated":"2021-06-24 11:13:08",
"OnCallDuration":25,
"Uri":"/v1/Accounts/exotelt/Calls/XXXXXXXXXXXXX/Legs/XXXXXXXXXXXXX"
},
{
"Sid":"XXXXXXXXXXXXX",
"CallSid":"XXXXXXXXXXXXX",
"AccountSid":"exotelt",
"PhoneNumber":"XXXXXXXXXXXXX",
"Status":"completed",
"Origin":"outbound-dial",
"LastAction":"",
"DateCreated":"2021-06-24 11:12:48",
```

```
"DateUpdated":"2021-06-24 11:12:57",
"OnCallDuration":5,
"Uri":"/v1/Accounts/exotelt/Calls/XXXXXXXXXXXXXX/Legs/XXXXXXXXXXXXXX"
},
{
"Sid":"XXXXXXXXXXXXX",
"CallSid":"XXXXXXXXXXXXXX",
"AccountSid":"exotelt",
"PhoneNumber":"XXXXXXXXXXXXX",
"Status":"completed",
"Origin":"outbound-dial",
"LastAction":"",
"DateCreated":"2021-06-24 11:12:58",
"DateUpdated":"2021-06-24 11:13:07",
"StartTime":"2021-06-24 11:12:58",
"EndTime":"2021-06-24 11:13:08",
"ExoPhone":"XXXXXXXXXX",
"BackupEcoPhone":"XXXXXXXXXX",

"OnCallDuration":4,
"Uri":"/v1/Accounts/exotelt/Calls/XXXXXXXXXXXXXX/Legs/XXXXXXXXXXXXXX"
}
]
}
```


The "Status" parameter can take one of the following values:

- queued
- in-progress
- completed
- failed

- busy
- no-answer

The "Origin" parameter can take one of the following values:

- **Inbound:-** Customers dialing into Exotel Virtual number
- **Outbound:-** 1st leg of connect API
- **Outbound-dial:-** Subsequent dial out from Exotel platform post 1st leg is established ( either via inbound/outbound)
- **Monitor:-** Supervisor leg performing the Listen / Whisper / Barge actions

1.

**Response Body on failure:**

JSON

```
{
"RestException":{
"Code":400,
"Message":"Bad Request",
"Description":"Please check the callSID passed"
}
}
```

**Rate Limit:**

This API is rate limited to 200 queries per minute. Once this limit has been crossed, your requests will be rejected with an HTTP 429 'Too Many Requests' code.

# 5.6. Play dynamic text or audio from URL in connect applet

**What is usual?**

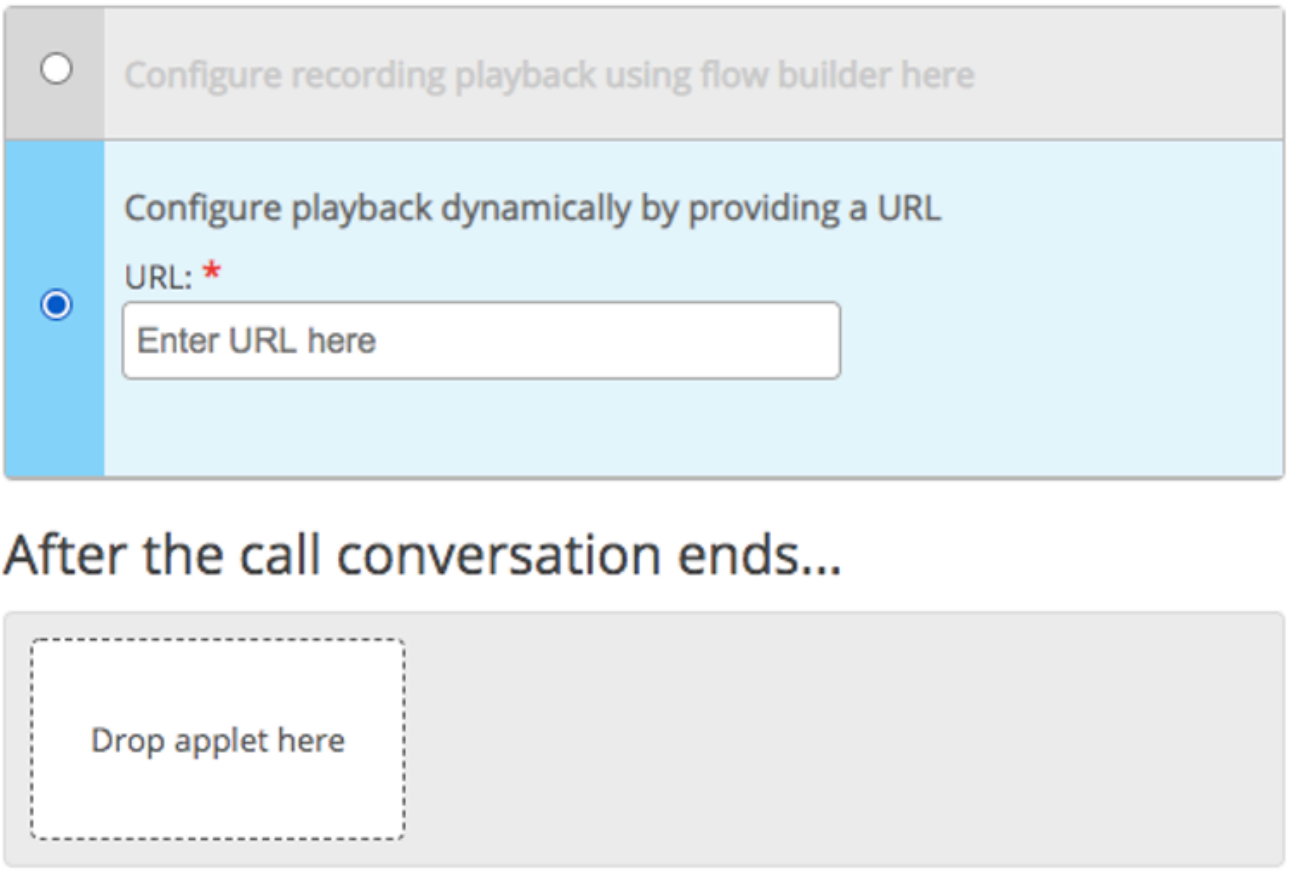
Typically, while configuring your Connect applet, you would enter the text or upload the audio that you want your agents to listen before the call gets connected to the customer. This is static in nature - ie, all your agents will hear the same message / audio.

**How to make it dynamic?**

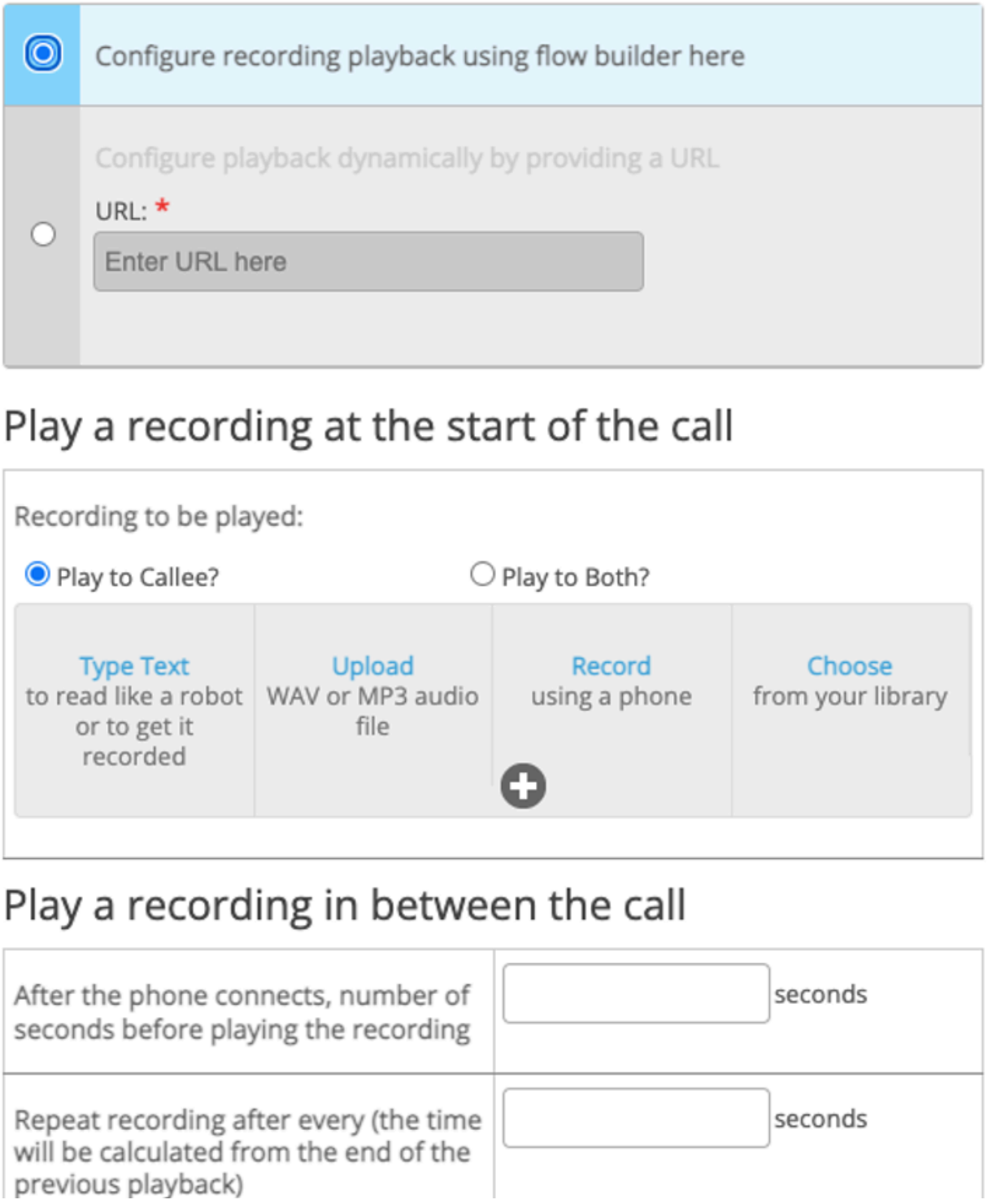
In Exotel, you have an option to make this dynamic - ie, you can potentially play a different greeting to each agent.

In the Connect applet, you can specify a URL (instead of a static text) in the "Read text like a robot" option. See below.

Choose this option if you want to control playback for each user uniquely




Choose this option if you want to have a static playback




When the option is chosen for URL, Exotel will make an HTTP GET request to that URL (which is hosted at your server). Your server can now return either a text or a URL (to an audio file). If it returns a text, the text will be converted to audio using our Text-To-Speech (TTS) engine and then played out.

The applets mentioned above are a part of the Custom App section also called App Builder, to know more about App Builder please follow----> **Exotel App Builder**

**HTTP Request from Exotel (to your URL)**

The GET request that Exotel makes to the URL will have the following query parameters:

| PARAMETER NAME | VALUE |
|---|---|
| CallSid | string; unique identifier of the call |
| From | string; the number of the calling party |
| To | string; your Exotel company number that is being called; this will be from your "Company Numbers" page |
| DialWhomNumber | string; the number that is being called currently (This might be empty also) |

**HTTP Response from your web server**

Your webserver

- MUST set the **Content-Type** HTTP header to '**application/JSON** and nothing else.
- MUST support a HEAD request from Exotel and return the exact same headers that it would for a GET request.
- The response will look something like

JSON

```
{
"start_call_playback":{
"playback_to":"both",
"type":"audio_url",
"value":"http://...mp3..."
OR
"playback_to":"both",
"type": “text”,
"value": "hello, this is a sample text"
OR
"playback_to":"callee",
"type":"audio_url",
"value":"http://...mp3..."
                        OR
               "playback_to":"callee",
               "type": "text",
                         "value": "hello, this is a sample text"
}
“in_call_playback:{
“start_delay”:5,
“repeat_frequency”:10,
“play_to_callee”:true,
“play_to_caller”:false,
“type”: “audio_url”,
“value”: “http://...mp3”
OR
“type”: “text”,
“value”: “hello, this is a sample text”
}
       “end_call_playback”:{
“type”: “audio_url”,
“value”: “http://...mp3”
OR
“type”: “text”,
“value”: “Hello, this is a sample text”
}
}
```

One can choose to give only **start_call_playback** or **in_call_playback** or **end_call_playback** and accordingly the audio will be played to users

NOTE

1. If the customer backend fails to respond to the GET request and Exotel receives 4xx or 5xx responses, then the audio will not be played to the user, and Connect applet will proceed and patch the call between callee and caller
2. The Audio files **MUST** be in the .wav/.mp3, 8Khz Mono format. The bit depth must be 16-bit.

NOTE: All other formats are **not** supported (like 16Kz Mono/Stereo etc)

**Caching of the audio files**

Exotel caches the audio files that it downloads and plays - so that the next time you send the same file, we don't need to download and it will be faster.

Exotel caches the file based on the **name of the URL.** So in case you are changing first-audio.wav, you will need to send in another name the next time (Ex: http://example.com/first-audio-new.wav)

# 5.7. How does the Campaigns API work

Before you start a Call Campaign, set a clear objective. We have some use cases here and guidelines here. Based on the business objective, configure a call flow. Campaigns API attempts to make calls to comma-separated ‘From’ numbers specified in the request. When a call is picked up, the flow is executed and status reported. A Campaign schedule determines when to start and end the campaign. Retry logic determines how and when to attempt to call a phone number again in case of failure.

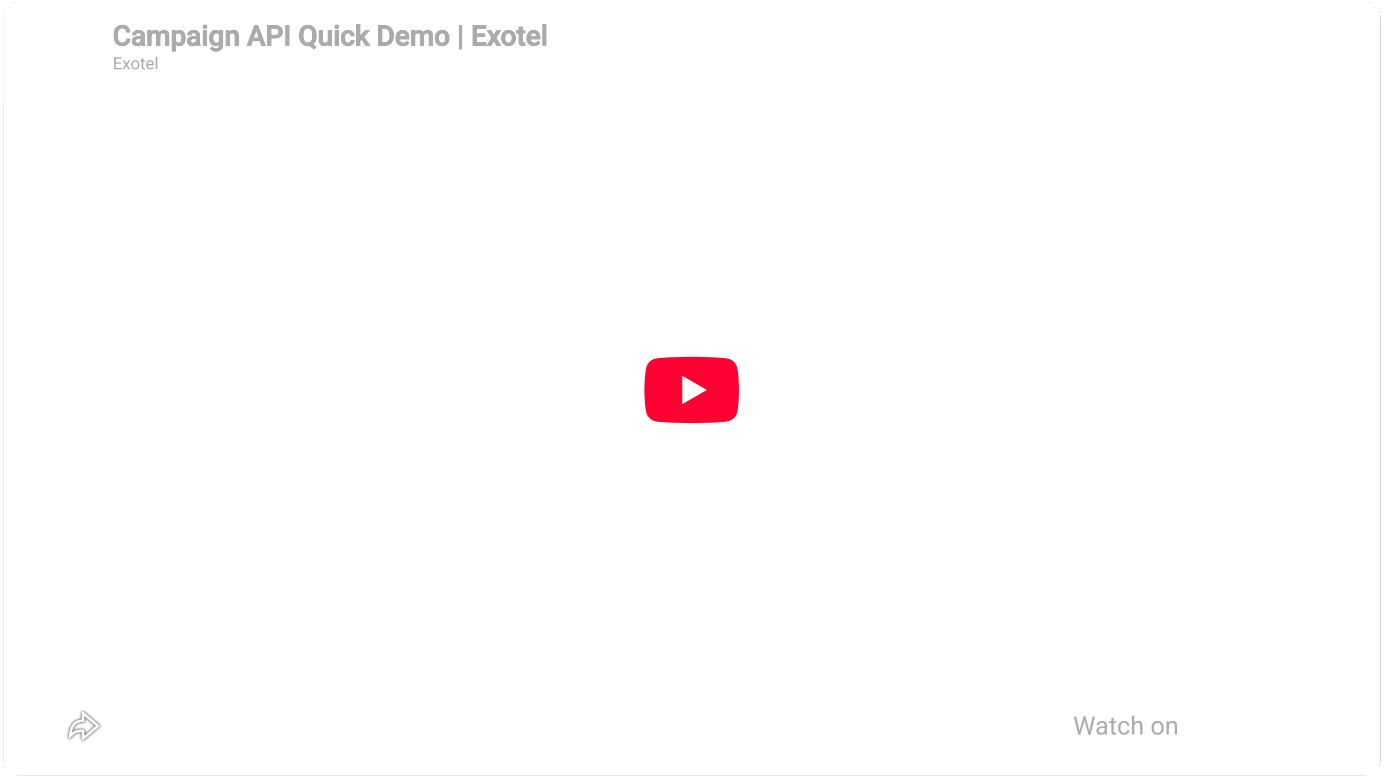
To use Campaigns API, including sample code, please refer to our developer portal

Below are some important considerations to keep in mind while using the API.

- When a request has an invalid number, then the entire request is failed with the number available in the failure message.
- The request will fail on the first invalid number found, the rest of the data is not processed. The developer is encouraged to use the phone number lib to validate his data before sending it.
- Terminal states of a campaign are ‘completed’, ‘failed’, ’deleted’, and 'canceled'. Intermediate states are 'created' , 'in-progress' and 'paused'.
    - Campaign States (intermediate)
        - **created** - A campaign API request that was successfully accepted by the system.
        - **in-progress** - A campaign that has started.
        - **paused** - A campaign, that was scheduled was requested to be paused explicitly.
    - Campaign States (Terminal):
        - **failed** - None of the calls were made (5xx /4xx / invalid numbers) for all of the phone numbers for the campaign. Immediate failures are marked as failed as well with the HTTP status code reflecting the reason for failure.
        - **completed** - all / partial calls (where the end time was reached) were made successfully.
        - **canceled** - A campaign was ‘paused’ but was not resumed ever and end_at was reached is marked as canceled. All the remaining calls for the campaign are marked as failed.
        - **deleted** - A campaign can be marked deleted only if it has not started.
- Call status
    - **in-progress** - When the call is triggered
    - **failed** - no retry was allowed and the status was ‘no-answer’, ‘busy’ or ‘failed’ state
    - **completed** - When this call was ‘completed’ by Exotel
- CustomFields are part of the report. CustomFields value is applicable to all numbers uniformly.
- Campaigns are marked as ‘failed’ if all calls were not attempted. If all calls were attempted, then that call campaign is successful, however, the status of individual calls will determine the success/failure of each call.
- A call to a number that failed (including retries) is marked as “failed”
- DELETE is possible only on campaigns that have not yet started.
- A campaign that is in progress, the below fields can be updated
    - Callbacks
    - Action
- When end_at is specified, Exotel makes its best effort to ensure that calls are completed within that period but is not guaranteed. POST request with end_at will return 200 OK irrespective of whether all calls can be made or not. All numbers that were not called because of timer expiry will be marked as ‘canceled’

- The report captures numbers that were not dialed for a failed campaign as well
- Long-Running Campaigns:
  - A cut-off time, 10.30 am - 9 pm is provisioned, which will imply that no call is made beyond the cut-off time.
- Paused campaigns without an end time specified will expire after 30 days from the start_at time.
- Throttle limits - 200 API requests per min with 60 concurrent channels

## Watch a short video



# 5.8. Why does the response time of Connect API vary?

***About the API:***

- Calls/connect API (POST) is used to connect two numbers or connect a number to a call flow.
- It connects the 'From' number first. Once the person at the 'From' end ( Leg 1 ) picks up the call, it connects to the number provided as 'To' ( Leg 2 ) or the flow (using 'Url').

For more information regarding the API specification, refer here.

***What is the API behaviour?***

Once Exotel receives an API request (**POST**) on Calls/connect, the platform validates the request. Post validation, Exotel initiates leg 1 (connecting the 'From' number in the API request) of the call by sending the request to the operator switch for call origination (which indicates that we have submitted the request to the operator for initiating the call and does not mean that it is ringing at this stage). Exotel also retries if the call origination fails. All of these actions are performed synchronously.

Upon successful origination of leg 1 of the call, a 200OK HTTP response is returned.

In an event where Exotel is unable to originate the call across any route, 500 Internal Server Error or 504 Gateway Timeout error is returned.

***Why is it synchronous?***

This is based on the philosophy that the inability to even originate a call is an immediate failure that the caller needs to know right away to enable other workflows.

For instance, if a delivery agent makes a request for an outbound call, the agent would like to know if the call was placed or not.

The analogy to this API request is similar to how you make a phone call from your handset, where you would like to see if the call originated or not as you hit the 'call' button. If the call origination fails, you will see it display on your handset immediately. There are

scenarios when we attempt to make a call from our handset and the call does not go through, then, our tendency is to retry manually. However, In the case of this API, we do the heavy lifting so that your call is connected.

Such reliability measures ensure call connectivity is higher and clients can take action based on the API responses.

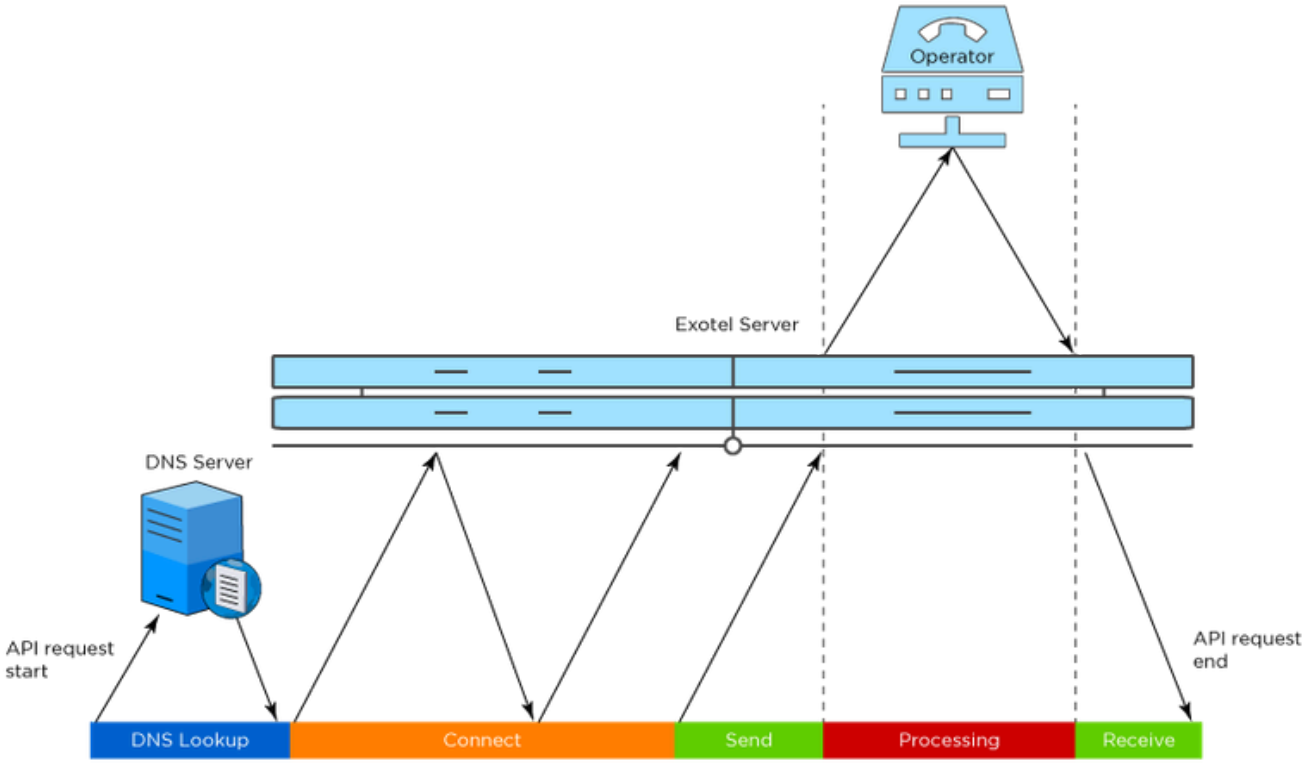
***What API response time to expect?***

'Response time' here is the total time taken from the instance API request is triggered from your system to Exotel and your system receives a response back to that API request.

The response time (as shown in the figure below) primarily constitutes:

- DNS lookup time
- Connection Time
- Time-taken by the client to send the request and receive the response
- Processing time taken by the server (elaborated further)
- Network latency is involved at each step

By response time, we are referring to the processing time of our platform. Other factors are beyond the scope of our platform and can vary based on the location of the requesting server, network connectivity, internet lines etc..




Historically*, the API latency for Calls/connect API observed is as follows:

| | |
|---|---|
| **Average API latency (in last 6 months)** | **~480 ms** |
| **95p API Latency (in last 6 months)** | **~1.1 seconds** |
| **99p API Latency (in last 6 months)** | **~2.5 seconds** |

**as calculated on 22nd October 2018*

***Note***: Exotel may continue to originate a call, for up to 30 seconds in the worst case.

Based on the above historical trends and behaviour, you can choose to set an appropriate request timeout in your application (this can vary as per your use case, implementation, business criticality etc). However, even if a request is timed out from the client side, our platform may still originate the call as the request might have already been forwarded to the operator switch.

***What can cause delays in API response?***

As described in the behaviour aspect of the API, Exotel will attempt to originate the call with the operator upon an API request and only then respond back (and not respond immediately upon receiving the request).

Few reasons which can cause a delay in originating the call and hence API latencies are:

- Network latency
- Operator switch takes longer to respond back
- Transient operator/network errors

Exotel has services in place which handle these scenarios by trying alternate routes which may add to the latency but ensure call connectivity is maintained. It's a tradeoff given the unreliable nature of operator networks.

***Best Practices:***

- Applications consuming this API are advised to handle the response internally in an asynchronous or non-blocking manner. This will depend on the implementation and programming language used.
- Set connection timeout and request timeout separately in your application. The time taken to establish a connection from an application client to Exotel will primarily depend on the network and location of your server. The request timeout can be set as per the API behaviour and response time trends detailed above.

# 5.9. How to monitor the status of your SMS?

For an SMS sent via Exotel SMS API, you can find out the delivery status of the SMS via two means:

**A. PULL:** You can pull the status by querying Exotel for the specific SMS that you sent.

For this you need to make a GET request to SMS API

https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/<ExotelSid>/SMS/Messages/<SmsSid>

- Replace <your_api_key> and <your_api_token> with the API key and token created by you. This can be found here: https://my.exotel.com/apisettings/site#api-credentials
- Replace <your_sid> with your “Account sid”
- Replace <subdomain>with the region of your account
    - <subdomain> of Singapore cluster is api.exotel.com
    - <subdomain> of Mumbai cluster is api.in.exotel.com

Ex: curl "https://asd23df:abcdefgh@api.exotel.com/v1/Accounts/Exotel/SMS/Messages/4dd44cc0f7010ee43ca256126f3efc88"

**B. PUSH:** You can have Exotel push the SMS status to you after the SMS reaches any terminal state (ie, "sent", "failed" or "failed-dnd").

For this, you need to pass an additional parameter "StatusCallback" along with your other parameters (like 'From', 'To' and 'Body').

=> This parameter should be a URL that is hosted by you. Ex: http://example.com/

=> Exotel will make a POST request to the above URL with the following parameters:

- SmsSid - The Sid (unique id) of the SMS that you got in response to your request
- To - Mobile number to which SMS was sent
- Status - one of queued, sending, submitted, sent, failed-dnd, failed
- SmsUnits - The number of SMS units being sent
- DetailedStatus - Human readable word that explains what happened to the message
- DetailedStatusCode - Exotel’s Detailed Status code corresponding to the DetailedStatus
- DateSent - The date on which the message was sent
- CustomField - The custom field that was set in the POST request. (Will be returned only if it was set)

**DASHBOARD:** You can also monitor the status of your SMS from the Outbox section of the dashboard.

| Detailed Status |
| --- |
| FAILED_SUBSCRIBER_ERROR |
| DELIVERED_TO_HANDSET |
| DELIVERED_TO_HANDSET |
| DELIVERED_TO_HANDSET |

Also, refer to How to check SMS status from the dashboard

**SMS STATS:** The SMS Stats dashboard gives you a high-level overview of your SMS stats.

The dashboard is accessible as SMS Stats (Beta) under the SMS section of your Exotel portal.

# 5.10. How do I integrate Exotel into my CRM?

Exotel is a cloud-based telephony system i.e. all of Exotel data is available over the web. This makes it cross-linkable with other cloud-based applications easily such as CRM, ERP, CMS, etc. There are 3 broad classifications of API implementations:

1. Incoming Calls/SMS via Passthru:
   The Passthru Applet makes an HTTP GET request onto a URL (on your server) with the details of the call. Every call accumulates data as the call progresses. For example, at the beginning of a call, the only data available is "From Number", "Time of Call" & "Call SID". As the call progresses, more information gets added to this call such as "To Number", "End Time", "Duration", "Recording URL". The fundamental principle of these integrations is to be able to pass on the call details onto your CRM to make an Auto-Entry. You can also use the "Agent Passthrough" to identify which agent is getting the call if the implementation involves popping-up the details of the customer on your CRM page.
2. Call API:
   The Call API is an HTTP-based REST API that allows you to trigger an outbound call using a button on your internal application. You can make use of the "Status Callback" to auto-update the details once the call has completed. You can even use the Call API to trigger IVR calls (calls from machine to human) and ask for a set of responses.
3. SMS API:
   The SMS API is also an HTTP-based REST API that allows you to trigger SMSs using actions/buttons at your internal application.

A combination of these 3 will give you extensive integration between Exotel and your CRM application. As the APIs are HTTP REST-based, Exotel can be integrated with any cloud-based application. Having said this, we offer native integration with the leading CRMs like the one listed below:

1. Salesforce

2. Zoho CRM

3. Zoho Desk

4. Zoho Bigin

5. Zoho One

6. Freshdesk

7. Freshworks CRM (Fresh Sales)

8. HubSpot

9. LeadSquared

# 5.11. Passing a Custom field in API

```
Question: Customer calls on my virtual number from Exotel and Exotel lets me know via callback that I have indeed received a call from this number. For this I am using miss called plus app. However, I was wondering if its possible for me to pass some custom fields per mobile number which you can then send it back to me as part of the callback in passthru applet.
```

Unfortunately, It is not possible to pass along your own parameter for incoming calls. But in most cases, we have found that it is possible to maintain the necessary custom variables & their values at your end. Because the From & To number is unique for a phone call ( You can also include a timestamp if needed or use the CallSid field that you get in the pings ), you could persist the custom values at your end and load them as soon as you get the ping from Exotel.

Passing a custom variable is possible for outbound calls via API. See the CustomField Parameter in our call connect API. We have listed all our APIs here **https://developer.exotel.com/api#call**

# 5.12. Metadata of a phone number

The number metadata API provides the following metadata of a given number:

- Telecom Circle
- Human readable Telecom Circle name
- Number type
- Operator (Carrier) that the number belongs to
- Whether the number is under DND or not

If all you want is to see/read the details of a phone number, use our **Phone number details page**

**Request Format:**

```
GET https://<your_api_key>:<your_api_token><subdomain>/v1/Accounts/<your_sid>/Numbers/~number~
```

Copy

https://<your_api_key>:<your_api_token><subdomain>/v1/Accounts/<your_sid>/Numbers/~number~

- Replace <your_api_key> and <your_api_token> with the API key and token created by you.
- Replace <your_sid> with your “Account sid”
- Replace <subdomain>with the region of your account
  - <subdomain> of Singapore cluster is api.exotel.com
  - <subdomain> of Mumbai cluster is api.in.exotel.com

<your_api_key> , <your_api_token> and <your_sid>onExotel Dashboard

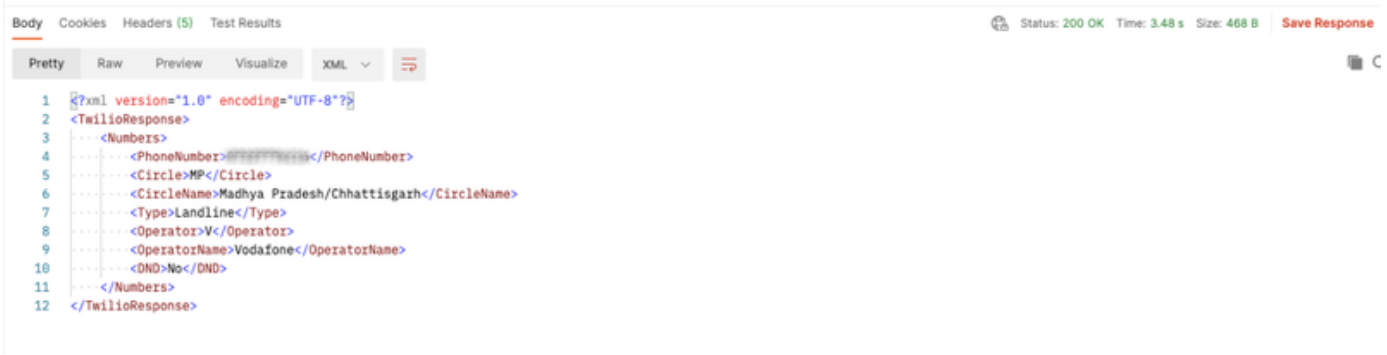
**Example:https://roopit:afancytokenwordhere007@api.exotel.com/v1/Accounts/roopit/Numbers/9916766748**

**Response:**

The response comes back in XML

| Parameter Name | VALUE |
|---|---|
| PhoneNumber | Reformatted phone number string |
| Circle | Two-character **telecom circle code**. |
| CircleName | The human-readable string of the Circle code.<br>Ex: Haryana Telecom Circle (excludes Faridabad, Gurgaon & Panchkula) |
| Type | 'Mobile' or 'Landline' |
| Operator & OperatorName | "AC" => "Aircel","A" => "Airtel","AL" => "Allianz","B" => "BSNL","D" => "Dishnet","E" => "Etisalat","H" => "HFCL","I" => "Idea","LO" => "Loop","MT" => "MTNL","P" => "Ping","R" => "Reliance","S" => "Sistema","ST" => "S Tel","T" => "Tata","U" => "Unitech","VI" => "Videocon","V" => "Vodafone" |
| DND | 'Yes'<br>'No'<br>'Unavailable' |

**Example:**



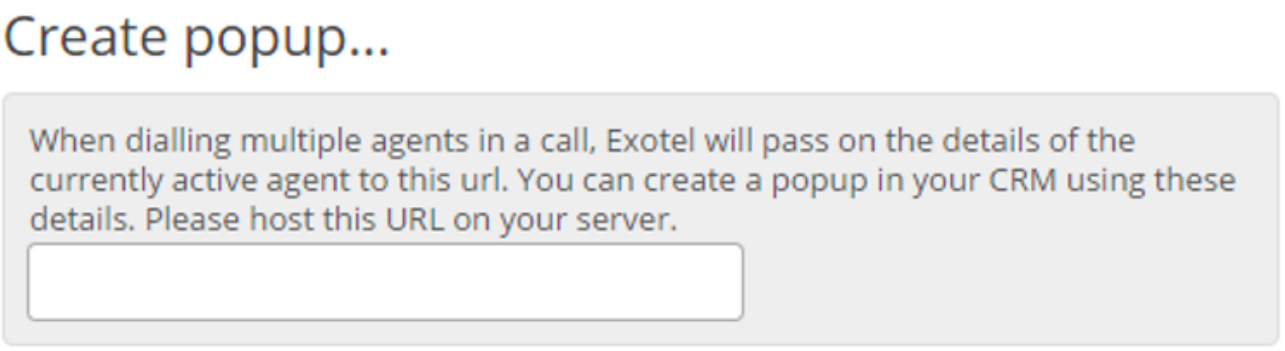

**Please note the DND status fetched using the mentioned API may not be accurate as the DND statuses are not published as per recent updates by TRAI.**

# 5.13. How can I get data of a call into another system using Exotel API?

There are two ways to monitor incoming calls in real time without using reports at all:

1. Use the Passthru applet - You can put the Passthru applet at the end of every incoming flow for all your shifts. This applet will basically transfer all call details (who called, who picked up, date, time, etc.) to a URL hosted on your CRM. Please add the URL in the Passthru applet and the Exotel system will make a GET request to that URL with the encoded call parameters and you'd be able to receive them at your end.
   This way, you will get all the call details in real-time on your CRM.


2. Use Agent Popup - this will help you monitor which agent picked up which call at what time. As soon as an agent picks up a call, the Exotel system will make a GET request to a URL provided by you and transfer the call details (caller's number, time, date, and the agent's number who picked up). This will be received on your end. This will be done in real-time, as soon as the agent picks up a call, with no delay whatsoever. It will be like a pop-up appearing on your CRM.
   The URL will be entered in a box on the CONNECT applet under the heading "During the Call".



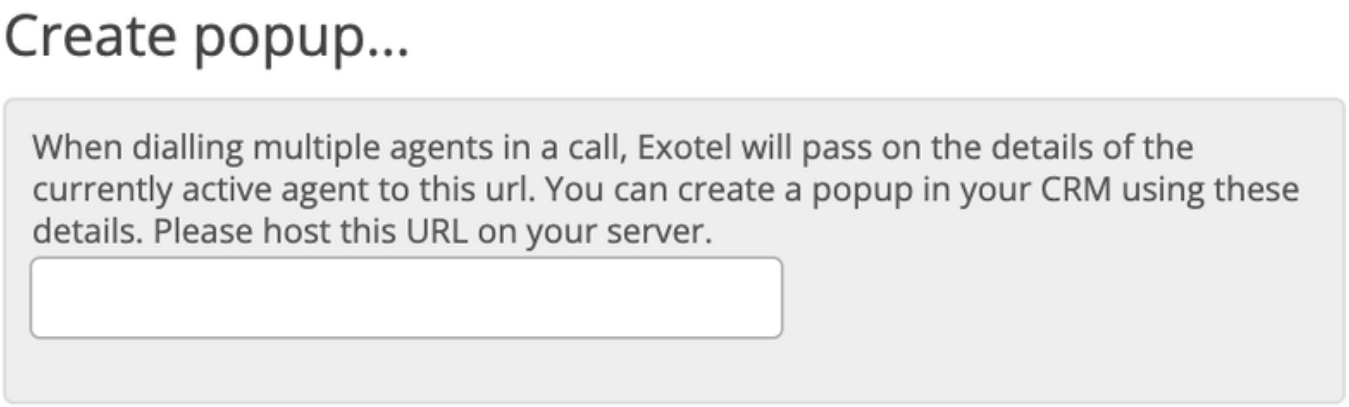

3. Use **Status Callback**- this works for outbound API calls. The outbound calling API has a parameter called StatusCallback where you can pass your endpoint URL hosted by your CRM and after the call is finished an HTTP POST request will be made to the endpoint URL that you have passed in the StatusCallback parameter and call details (caller's number, time, date, and the agent's number who picked upand custom filed can also be added) will be transferred to your CRM.

Please click here to know more about the StatusCallback parameter.

# 5.14. Getting a Popup when agent receives a call

To create a Popup for an agent when he receives a call, this needs to be configured in connect applet in the call flow. In the connect applet, you need to go to the section titled "Create popup...", shown below.

Here, enter the endpoint URL where the data has to be pushed to create the popup.




When configured with a URL, Exotel will pass on to that URL details of the agent who is currently being called. It makes a GET request to the URL with the following query parameters:

| PARAMETER NAME | VALUE |
|---|---|
| CallSid | string; unique identification for every single call |
| From | string; the number of the caller (customer in this case) |
| DialWhomNumber | string, the number of the agent |
| Status | busy: Status is sent as 'busy' when the above agent is being called.<br>free: Status is sent as 'free' when the call to the above agent ends. The free status will also be triggered, when the call ends or the call didn't go through or the agent didn't pick up the call. |
| EventType | string, This shall have values of '' Dial'', ''Ringing'', ''Answered'', ''Terminal'' |

NOTE:-

- To get a Ringing event or Answered event in the connect applet, it needs to be enabled by Exotel for your account
- The ringing event is dependent on the operator and it may not be reported if the operator doesn't provide it in certain cases. If the ringing event is not available then the RingDuration is set as 0.

# 5.15. Outbound Call to connect an Agent to a Customer

This API will connect the two numbers given for an agent and the customer. It will connect the ‘From’ number first. Once the person at the ‘From’ end picks up the phone, it will connect to the number provided in the ‘To’. You can choose which number to be connected first by giving that number in the ‘From’ field.

An HTTP POST request to https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/<your_sid>/Calls/connect has to be made

- Replace <your_api_key> and <your_api_token> with the API key and token created by you.
- Replace <your_sid> with your “Account sid”
- Replace <subdomain>with the region of your account
  - <subdomain> of Singapore cluster is api.exotel.com
  - <subdomain> of Mumbai cluster is api.in.exotel.com

<your_api_key> , <your_api_token> and <your_sid> are available on the API settings page of your **Exotel Dashboard**

The following are the POST parameters:

| PARAMETER NAME | MANDATORY/OPTIONAL | VALUE |
|---|---|---|
| From | Mandatory | The Agent's phone # that will be called first<br>If mobile number, prefix the 10 digits with a 0; Ex: ✆✆ 09052161119.<br>If landline number, prefix it with STD code; Ex: 08030752400 |
| To | Mandatory | Your customer's phone number.<br>If mobile number, prefix the 10 digits with a 0; Ex:✆✆ 09052161119.<br>If landline number, prefix it with STD code; Ex: 08030752400 |
| CallerId | Mandatory | This is your Exotel Number (pick one from the 'Company Numbers' page) |
| CallType | Mandatory | |
| TimeLimit | Optional | The time limit (in seconds) that you want this call to last. The call will be cut after this time. (max. 14400 i.e. 4 hours) |
| TimeOut | Optional | The time (in seconds) to ring the called parties (both first and second call leg) |
| StatusCallback | Optional | When the call completes, an HTTP POST will be made to the URL mentioned with the following four parameters:<br>CallSid - an alpha-numeric unique identifier<br>Status - One of completed, failed, busy, no-answer.<br>RecordingUrl - link to the call recording (if it exists)<br>DateUpdated - time when the call state was updated last |
| MaxRetries | Optional | The number of times the call should be retried if we get failed or no-answer status from 1st leg.<br>(Default value: 3) |

"trans" - for Transactional Calls
~~"promo" - for Promotional calls~~ No longer supported by Exotel

**HTTP Response:**

**On success,**

- the HTTP response status code will be 200
- the HTTP body will contain an XML similar to the one below. The "Sid" is the unique identifier of the call and it will be useful to log this for future debugging purposes.

```
<?xml version="1.0" encoding="UTF-8"?>

<TwilioResponse>
<Call>

<Sid>xxxxxxxxxxxxxxxxxxxx</Sid>
<ParentCallSid/>

 <DateCreated>2012-08-17 12:31:49</DateCreated>

 <DateUpdated>2012-08-17 12:31:49</DateUpdated>

 <AccountSid>xxxxxxxxx</AccountSid>

 <To>09052161119</To>

 <From>09739761117</From>

 <PhoneNumberSid>xxxxxxx</PhoneNumberSid>

 <Status>in-progress</Status>

 <StartTime>2012-08-17 12:31:49</StartTime>

 <EndTime>2012-08-17 12:32:57</EndTime>

 <Duration></Duration>

 <Price></Price>

 <Direction>outbound-api</Direction>

 <AnsweredBy/>

 <ForwardedFrom/>

<CallerName/>
<RecordingUrl/>

 <Uri>/v1/Accounts/xxxxxxxx/Calls/xxxxxxxxxxxxxx</Uri>

 </Call>

</TwilioResponse>
```

**On failure,**

- the HTTP response status code will be non-200.
- the HTTP body will contain an XML (such as the one below) with details of why the request failed.

```
<TwilioResponse>
<RestException>
<Status>404</Status>
<Message>No matching results</Message>
</RestException>
</TwilioResponse>
```

**Rate Limit:**

This API is rate limited to 200 calls per minute. Once this limit has been crossed, your requests will be rejected with an HTTP 429 'Too Many Requests' code.

*Sample PHP Code:*

An **example PHP code for this is available on Github**

*Sample Ruby Code:*

Sample code is available at **Github** and Ruby Gem available at **rubygems.org**

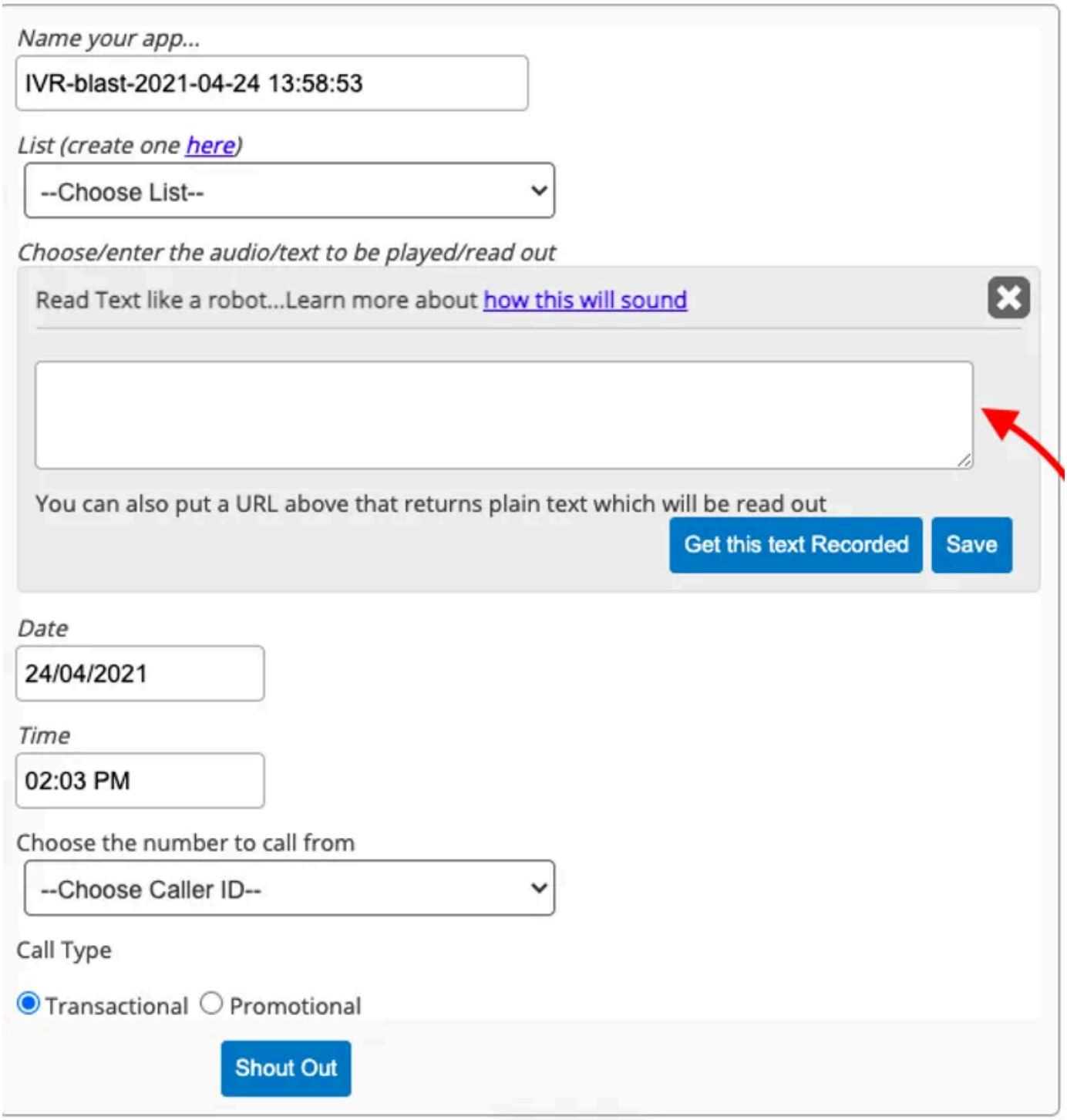
## Watch a short video

# 5.16. API to convert text to high quality voice using Shout Out app

*Question*: How do I convert text to high-quality voice and deliver it to cell phones? Can you provide me with an API for such a task? These text messages would be different for different customers.

*Solution*: You may use our "**Shout Out**" app to create bulk IVR call campaigns in minutes. Simply choose a list and the message to play out and schedule the time to Shout Out!

*Here's how*: Download the Shout Out app from the app store. Put a URL that can return text in the text box. The parameters for this URL will be the same as for the **Passthru applet**. Because one of the parameters is the phone number, your URL can return different text for different callers, and the content will be read out using our Text to Speech engine.




Read more about **greeting callers using dynamic text or audio**.

# 5.17. Outbound Call to connect a Customer to an App

This API will first call the customer, and once they pick up the phone, it will connect them to an app (aka flow) that you have created in the system - like your landing app, or any other app that can play a greeting, have IVR etc.

A HTTP POST request to https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/<your_sid>/Calls/connect has to be made

- Replace <your_api_key> and <your_api_token> with the API key and token created by you.
- Replace <your_sid> with your “Account sid”
- Replace <subdomain>with the region of your account
  - <subdomain> of Singapore cluster is api.exotel.com
  - <subdomain> of Mumbai cluster is api.in.exotel.com

<your_api_key> , <your_api_token> and <your_sid> are available in the API settings page of your **Exotel Dashboard**

The following are the POST parameters:

| PARAMETER NAME | MANDATORY /OPTIONAL | VALUE |
|---|---|---|
| From | Mandatory | The customer phone # that will be called first<br>In the case of mobile numbers, prefix the 10 digits with a 0; Ex: 09052161119.<br>In case of landline number, prefix it with STD code; Ex: 08030752400 |
| CallerId | Mandatory | This is your Exotel Number (pick one from the 'Company Numbers' page) |
| CallType | Mandatory | "trans" - for Transactional Calls |
| Url | Mandatory | "http://my.exotel.in/exoml/start/ ~*appid*~"<br>where ~*appid*~ is the identifier of the app (or flow) that you want to connect to once the 'From' number picks up the call |
| TimeLimit | Optional | The time limit (in seconds) that you want this call to last. The maximum is 4 hours. The call will be cut after this time. |
| TimeOut | Optional | The time (in seconds) to ring the called parties (both first and second call leg). Maximum is 1 minute (which, we hear is mandated by TRAI) |
| StatusCallback | Optional | When the call completes, an HTTP POST will be made to the URL mentioned with the following four parameters **(This data will be passed in the Message body and as multipart/form-data)** :<br>CallSid - an alpha-numeric unique identifier<br>Status - {completed, failed, busy, no-answer}<br>RecordingUrl - link to the call recording (if it exists)<br>DateUpdated - time when the call state was updated last |
| CustomField | Optional | Any application-specific value that will be passed back as a parameter while doing a GET request to the URL mentioned in your **Passthru Applet** or **Greetings Applet**. |

**HTTP Response:**

**On success,**

- the HTTP response status code will be 200
- the HTTP body will contain an XML such as what's given below. The "Sid" is the unique identifier of the call and will be useful to log this for future debuggability purposes.

```
<?xml version="1.0" encoding="UTF-8"?>

<TwilioResponse>

<Call>

<Sid>xxxxxxxxxxxxxxxxxxxx</Sid>

<ParentCallSid/>

<DateCreated>2012-08-17 12:31:49</DateCreated>

<DateUpdated>2012-08-17 12:31:49</DateUpdated>

<AccountSid>xxxxxxxxx</AccountSid>

<To>09052161119</To>

<From>09739761117</From>

<PhoneNumberSid>xxxxxxx</PhoneNumberSid>

<Status>in-progress</Status>

<StartTime>2012-08-17 12:31:49</StartTime>

<EndTime>2012-08-17 12:32:57</EndTime>

<Duration></Duration>

<Price></Price>

<Direction>outbound-api</Direction>

<AnsweredBy/>

<ForwardedFrom/>

<CallerName/>

<RecordingUrl/>

<Uri>/v1/Accounts/xxxxxxxx/Calls/xxxxxxxxxxxxxx</Uri>

</Call>
```

```
</TwilioResponse>
```

**On failure:**

- the HTTP response status code will be non 200
- the HTTP body will contain an XML such as below with the details of why the request failed

```
<TwilioResponse>
<RestException>
<Status>400</Status>
<Message>Invalid Call Parameters: No 'From' specified</Message>
</RestException>
</TwilioResponse>
```

**Rate Limit:**

This API is rate limited to 200 calls per minute. Once this limit has been crossed, your requests will be rejected with an HTTP 429 'Too Many Requests' code.

You can run this API in cURL, NodeJS, PHP, Python, and Ruby. The requests can be taken from the **developer portal**

*Sample PHP Code:*

An **example PHP code for this is available on Github**

*Sample Ruby Code:*

Sample code is available at Github and Ruby Gem available at rubygems.org

## Watch a short video

**Connecting a customer to an app with an outbound call**

You are not permitted to perform this action

# 5.18. Call API to get details of a Call

To get details of a call (including Status, Price, etc.), you will need to make an HTTP GET request to

https://<your_api_key>:<your_api_token>@<subdomain>/v1/Accounts/~exotel_sid~/Calls/<CallSid>

Replace <your_api_key> and <your_api_token> with the API key and token created by you.

Replace <your_sid> with your “Account sid”

Replace <subdomain> with the region of your account

<subdomain> of Singapore cluster is @api.exotel.com

<subdomain> of Mumbai cluster is @api.in.exotel.com

<your_api_key>, <your_api_token> and <your_sid> are available on the API settings page of your Exotel Dashboard

In case of an outbound call originating via the Call APIs (here or here), the "Sid" parameter is as returned in the XML response to your request.

In case of inbound calls, you can get the CallSid by using the Passthru applet.

**HTTP Response:**

**On success**, this API will return an HTTP status code of 200 and the response body will contain an XML (like below) that contains the details of the call.

**On failure**, the HTTP status code will be a non-200 code which indicates the reason (as discussed here ).

**IMPORTANT NOTE:** Some of the parameters of the call (like Duration, Price, EndTime etc.) are updated asynchronously after the call ends. So it might take some time after the call ends (~ 5 mins on average) for these parameters to be populated correctly.

**Response Body on Success:**

<?xml version="1.0" encoding="UTF-8"?><TwilioResponse><Call><Sid>xxxxxxxxxxxxxxxxxxxx</Sid><ParentCallSid/>
<DateCreated>2012-08-17 12:31:49</DateCreated><DateUpdated>2012-08-17 12:31:49</DateUpdated>
<AccountSid>xxxxxxxxx</AccountSid><To>09052161119</To><From>09739761117</From>
<PhoneNumberSid>xxxxxxx</PhoneNumberSid><Status>completed</Status><StartTime>2012-08-17 12:31:49</StartTime>
<EndTime>2012-08-17 12:32:57</EndTime><Duration>123</Duration><Price>2.3</Price><Direction>outbound-api</Direction>
<AnsweredBy/><ForwardedFrom/><CallerName/><RecordingUrl/><Uri>/v1/Accounts/xxxxxxxx/Calls/xxxxxxxxxxxxxx</Uri></Call>
</TwilioResponse>

The "Status" parameter can take one of the following values:

- queued

- in-progress

- completed

- failed

- busy

- no-answer

**Response Body on failure:**<?xml version="1.0" encoding="UTF-8"?><TwilioResponse><RestException><Status>xxx</Status>
<Message>xxxxxxxxxxxxxxxxxxxx</Message></RestException></TwilioResponse>

**Rate Limit:**

This API is rate-limited to 200 queries per minute. Once this limit has been crossed, your requests will be rejected with an HTTP 429 'Too Many Requests' code.

A detailed explanation about using the call detail API can be understood from our **developer portal**.

# 6. IP-PSTN intermix: Customer onboarding and WebRTC SDK integration

## Overview

Exotel's WebRTC SDK provides an easy-to-integrate SoftPhone SDK for web applications. The SDK allows you to embed a fully functional softphone within your web applications which can be an in-house web portal or a CRM Application, enabling seamless communication between your users and customers. The softphone supports both inbound and outbound calling, along with programmatically making calls- via dial-pad in the widget and by clicking to call in your CRM. By following the simple integration steps provided, you can quickly enhance your applications with powerful voice communication capabilities.

Here's a short **video** displaying the WebRTC SDK capabilities:

#### Benefits of the WebRTC SDK

1. **Bulk user creation using a single API** - The users for IP calling can be created at once, with the help of bulk user creation API.
2. **SIP ID and password creation/management for the users**- The SIP credentials of the users would be created and maintained by Exotel. Users just need to log in to their CRMs and get started with the IP calling.
3. **SSO experience for the users -** Users just need to log in once to their CRMs and need not separately login to Exotel. On successful login to the CRMs, the users would be authenticated and the WebSDK would be initialised and registered for the IP calling.
4. **IP calling within your CRMs** - The users would be able to take the SIP calls from the widget embedded within the CRMs and need not toggle between multiple tabs.
5. **UI widget for calling**- The widget for incoming/outgoing call popup would be shown to the users where they would get options to accept/reject the calls along with the hold/mute buttons.
6. **The ability for the users to switch off/on their SIP device**- Users can switch between IP and PSTN calling as per their network availability with the help of a toggle button.
7. **On-call statuses, status callbacks, call details etc.**- All the call details would be available post each call for you to analyze the calls.

#### Onboarding process

Please follow the below-mentioned steps to use and configure the IP-PSTN WebRTC SDK for VoIP calls:

#### Pre-requisites

In order to have a successful integration with the IP-PSTN WebRTC SDK, you must complete the following prerequisites:

1. **New Services Agreement**
   To comply with the Government of India regulations, Exotel has obtained a Unified License for Virtual Network Operator (UL-VNO) under a separate entity named 'Veeno Communications Pvt. Ltd.' Therefore, a new Services Agreement needs to be signed with 'Veeno Communications Pvt. Ltd.' for using VoIP services. Your account manager or the Exotel point of contact will help you with

this step.

2. **New account creation (even for existing customers)**

Due to the regulatory reason/s mentioned in point 1 above, you are required to sign-up for a new account to use IP-PSTN intermixing for VoIP calls. VoIP services cannot be provided under the existing account/s which you may have already created with us. Your account manager or the Exotel point of contact will help you with this step.

3. **New KYC to be registered (Bangalore ONLY for now)**

- In addition to the four documents required to complete the KYC for an account, accounts with VOIP services enabled for them also require a Customer Acquisition Form(CAF) to be completed as part of KYC. It is supported on the Exotel dashboard as one of the document requirements on the KYC page.
- KYC needs to be done separately for each city for VoIP accounts. This primarily means that the following 2 documents need to be produced fresh every time a new city KYC is to be done:

1. Address proof (The address needs to be of the city from where the numbers need to be purchased)
2. Customer Acquisition Form (CAF)

- For more details on where the KYC/Verification docs need to be uploaded and what documents qualify for KYC, **this link** should be referred to. Please get in touch with your account manager or the Exotel point of contact for any queries regarding this process.

4. **New Exophones (Virtual Numbers) to be purchased**
   Virtual Number purchase is coupled with individual region/city KYC status. i.e. number purchases will only be allowed from regions where the KYC is completed.

NOTE: Currently, the only city available for KYC and number purchase is Bangalore

For more details on how to procure a new Exophone, please refer to **this support article**

5. **Call flows creation/migration and linking with Virtual Number/s**
   Once the new account is created and the virtual number/s is purchased, you would need to create the call flows for the inbound scenarios.
   Reference article - **Link**

   If you already have an account with Exotel with call flows created for the inbound scenarios and wish to use the same call flows in the new account for IP calling, the call flows would need to be migrated. Reference article - **Link**

https://support.exotel.com/support/solutions/articles/3000112569-can-i-transfer-my-exophone-and-the-call-flows-from-one-account-to-another-

Once the call flows are created/migrated, you can link the call flows to the virtual numbers so that any customer calls landing on those virtual numbers follow the required call flow path.

6. **Browser compatibility with the WebRTC SDK**

- Windows OS: Google Chrome, Mozilla Firefox and Microsoft Edge
- Linux OS: Google Chrome and Mozilla Firefox
- Mac OS: Google Chrome, Mozilla Firefox, Safari (>V11)

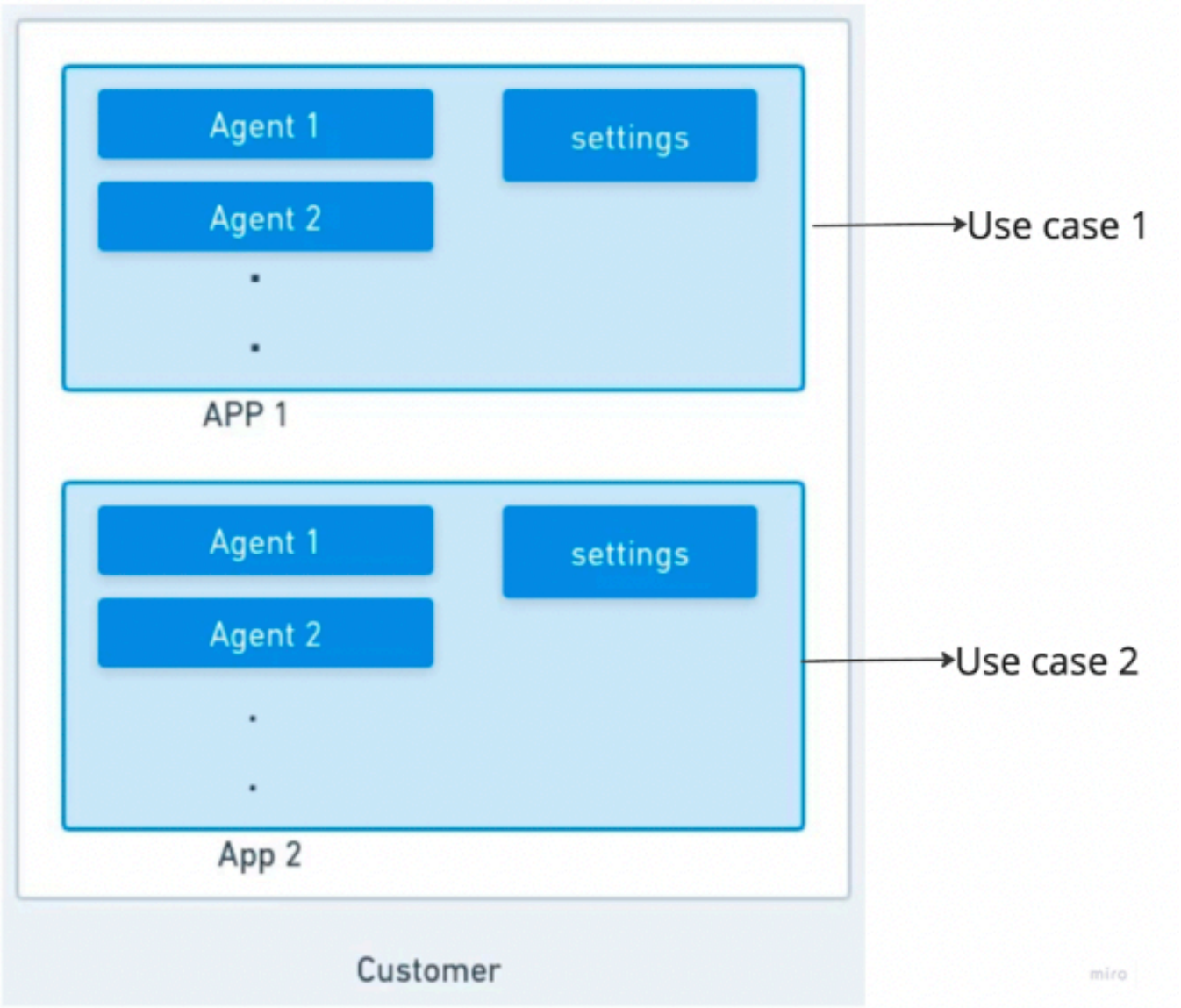
**Integration steps for the WebRTC SDK**

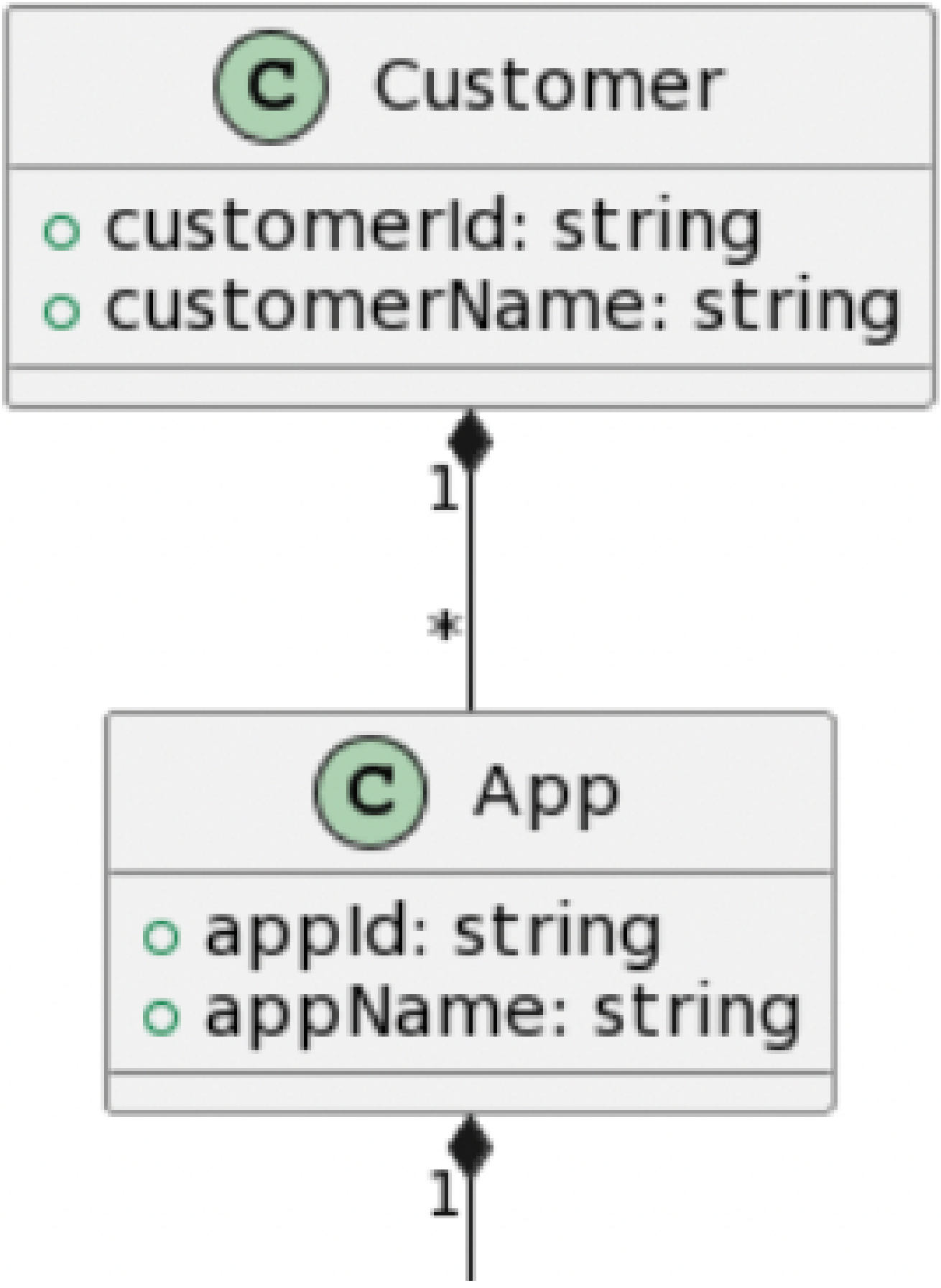
After the above-mentioned prerequisites are done, you would be able to fetch the API key and API token for your account, from the Exotel dashboard. In addition to this, you'll need a client id and client secret to generate authentication token/s for successfully authenticating your future API requests by our platform. Please reach out to the Exotel tech support team or your account manager for help in generating the client id and client secret.

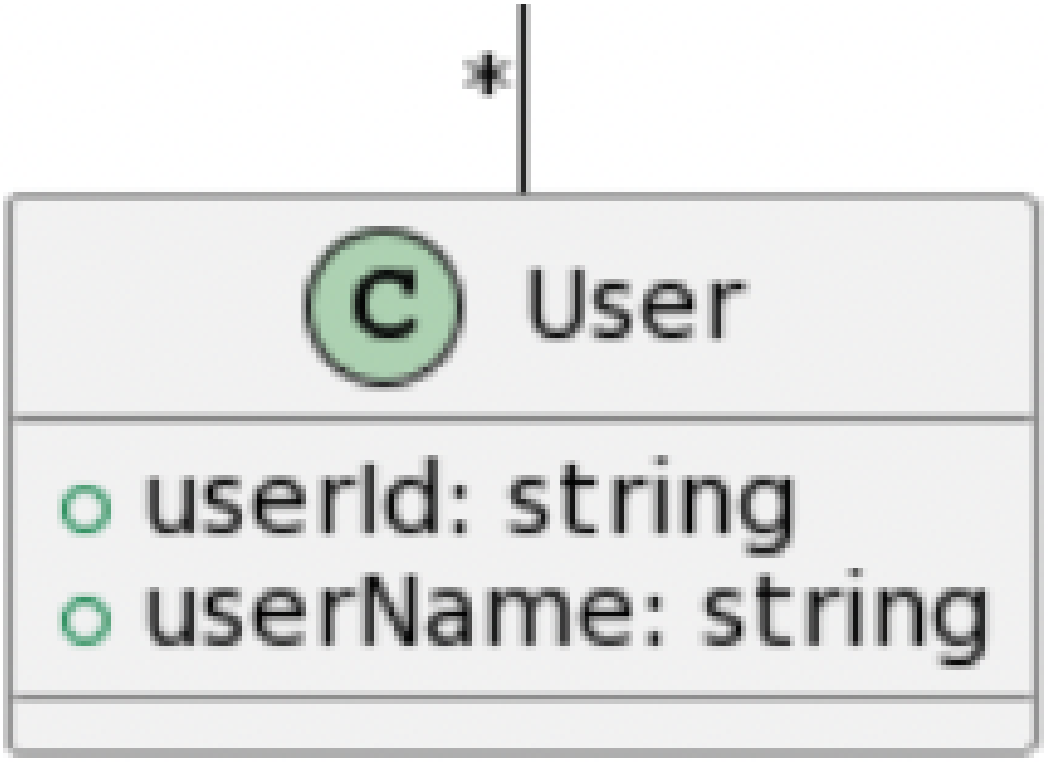
Once the client id and the client secret are available, please use the following API to generate the authentication token which can be used with your future API requests-

**Create Authentication Token**

The creation of the following resource hierarchy is required for registering with our VoIP platform to enable VoIP calling:




Customer
customerId: string
customerName: string
1
*
App
appId: string
appName: string
1




1. You are required to create a customer entity (steps mentioned below). A unique customer entity in Exotel's platform allows us to achieve resource separation between different customers.
    1. Use the following APIs to manage the Customer entity for VOIP calling**: Create, Get and Delete APIs for the Customer entity**
2. After step one, you will then need to create at least one, or more, Applications within the customer entity. Application creation is required to support multiple use cases within your single CRM/3rd party application account. ***For example, if your sales and support teams are mapped to a single CRM account which is mapped to a single Exotel account, but the users within these teams and their VOIP settings are to be maintained differently; then creating an Application for each of these teams would help you manage the VOIP calling requirements for both these teams separately using the same CRM account mapped to one Exotel account***. The creation of more than one Application is completely optional.
    1. Use the following APIs to manage Application(s) within the Customer entity**: Register, Get and Delete APIs for the App entity**
    2. Use the following APIs to manage Application settings**: Add, Get, and Delete APIs for App settings**
3. After step 2, you will need to add the user(s) within the Application(s). User addition through the Application results in the addition of these users into the Exotel system and the creation of their SIP credentials, which will be used to authenticate and verify a user while enabling their VoIP calling.
    1. Use the following APIs to manage Users within Application/s**: Register, Get, Delete, and Update user device APIs for Users**
4. After step 3, you will need to configure call flows in the Exotel App Bazaar as per Step 5 in the pre-requisites- 'Call flows creation/migration and linking with Virtual Number/s'. If you want Exotel to pass various details related to the incoming call when the call is initiated, terminated or missed, please provide the following URL in the 'Create popup...' section while configuring the 'Connect' applet during the call flow creation process:
    **https://integrationscore.mum1.exotel.com/v2/integrations/call/inbound_call/{appId}?type={}**

**Note**: The type can be popup, incomingcallhungup & missedcall, please refer to this detailed API documentation to understand. In case of multiple applications within your account, please use the application ID for the app under which you want the calling to happen.

5. Once the users are created, you will need to integrate the WebRTC SDK by following the steps from the following **GitHub repository**.
6. Finally, whitelist the following ports, domains and IP addresses in all the networks' firewall where the WebSDK will be running

| Type | Port/Endpoint |
|---|---|
| HTTP | tcp/80 |
| HTTPS / TLS | tcp/443 |
| SIP Gateway IP | voip.in1.exotel.com |
| Media Server Ports | udp/10000 - 40000 |
| Media Server IP - Bangalore, KA | 61.246.82.75, 14.194.10.247 |
| Media Server IP - Mumbai, MH | 182.76.143.61, 122.15.8.18 |
| HTTPS for API management | integrationscore.mum1.exotel.com |

7. Click to Call functionality is a part of the WebRTC SDK and will be functional on the integration of the SDK with your calling application.

**Pricing**

IP-PSTN intermixing is available only with an 'Unlimited Calling Plan' by paying a fixed per-user per month price. Please get in touch with your Exotel account manager to understand the pricing applicable to your requirements.

**FAQs**

1. **Do I need to sign a new MSA for VOIP calling?**

Yes. This is required because Exotel has a Pan India UL VNO license under the entity named "Veeno Communications Private Limited". Hence, there will be a new MSA to be signed between the customer and the Veeno entity.

2. **Do I need to create a new account for VOIP calling?**

Yes. All customers, including existing customers, will have to create a new account if they need VOIP calling. As VOIP calling is being offered through the Veeno entity, thus, a new customer account has to be created under Veeno as the existing account under Exotel Techcom Pvt. Ltd. cannot provide VOIP calls with the Veeno infrastructure.

3. **What is the process to create an account under Veeno?**

In the alpha phase, the account creation will be aided by the Exotel account management/sales team. Post alpha phase, the account creation for Veeno/VOIP calling will follow the same process followed today for non-VOIP accounts.

4. **Can I use the virtual numbers/Exophones that I have in my existing account for VOIP calling?**

No, new numbers have to be purchased under the new Veeno account. Existing numbers from non-Veeno accounts cannot be used under Veeno accounts because there is a separate number inventory for Veeno and it cannot be mixed with the non-Veeno number inventory as per the government regulations.

5. **If I have IVR/call flows in my existing account, do I need to create/replicate new IVR/call flows under the new Veeno account?**

The Exotel team will do everything possible to migrate the IVR/call flows from your existing account to the new Veeno account.

6. **What is the size of WebSDK?**

It is 1.4MB

7. **What would be the 'app user ID' for adding users in Exotel through the SDK?**

The email ID of the user should be used as the user id for the app while adding the users in Exotel through the SDK.

8. **Can the users be added via the Exotel dashboard?**

No, the users should be added only by following the steps mentioned for user creation in this support article. This process will ensure that the users' SIP credentials are created and managed successfully at our end.

9. **What are the prerequisites for the users to start calling via WebSDK?**

The users will need to log in to the third-party application with valid credentials. On successful login, the users will see their SIP device as switched on for making VOIP calls. They can toggle it off/on to switch between SIP and PSTN calling.

10. **How can the user stop making/receiving calls via WebSDK?**

The user will have to toggle off the SIP calling button
OR

The user will have to log off from the third-party application. This would automatically unregister and initialize the SDK for calling.

11. **What will happen if the user is not able to make/receive VoIP calls due to poor network connectivity?**

In such a case, the user can switch to PSTN calling with the help of a toggle button which means the mobile device of the user would become the primary device for calling

12. **How do I connect a user to a flow?**The WebSDK doesn't support dialling to a flow directly. This would require you to use the "Outgoing call to connect number to a call flow" API and pass the SIP ID in the From parameter and pass the flow link in the Url parameter.

# 7. Answering Machine Detection (AMD)

**Status:** Beta. Please reach out to your Exotel account manager to have AMD enabled on your account.

## Overview

Answering Machine Detection (AMD) tells you whether an outbound call was answered by a **human** or by a **machine** (such as voicemail or an IVR). This lets you take different actions depending on who picked up – for example, playing a message only when a human answers, or skipping voicemail drops.

AMD results are delivered through Exotel's call callbacks (APIs). They are **not** shown in reports or the dashboard.

## How AMD works

When AMD is enabled, Exotel analyzes the audio at the start of the call – the greeting and the silences around it – to classify who answered. The classification is returned through two new parameters inside the Legs object of your callback payload:

| Parameter | Description |
|---|---|
| AnsweredBy | The AMD result. Use this for your final classification. |
| AmdCause | The reason behind the result, including the values used for detection. Use this for debugging or understanding why a result was given. |

Possible AnsweredBy values

- HUMAN – the call was answered by a person
- MACHINE – the call was answered by a machine (voicemail, IVR, etc.)
- NOTSURE – detection could not reach a definitive result
- NA – AMD was not applicable for this leg

## Key behaviors

**1. AMD applies only to Leg 2 (the To leg).** Detection runs only on the number being dialled (Leg 2 / the To leg). Leg 1 (the From leg) will always return AnsweredBy: NA. When reading the payload, use the AnsweredBy value on Leg 2 for the human-vs-machine result.

**2. AMD detection is asynchronous.** Detection does not block or delay call connection or routing – the call proceeds normally while analysis happens in the background. Because detection listens to the greeting and surrounding silence, the result takes roughly **3–5 seconds** to finalize and is delivered once complete. Consume the result from the callback payload when it arrives, rather than expecting it the instant the call is answered.

## Where you receive the result

**A. StatusCallback terminal event**

To receive the AMD status via StatusCallback, you must **subscribe to terminal events**. Once subscribed, the AMD result is included in the terminal StatusCallback payload, inside the Legs object.

**Sample terminal StatusCallback payload:**

```
{
  "CallSid": "f3dec10279a310e2831754a4ec841a5f",
  "ConversationDuration": "13",
  "DateCreated": "2026-05-15 16:46:13",
  "DateUpdated": "2026-05-15 16:46:50",
  "DetailedStatus": "NA",
  "DetailedStatusSubBucket": "NA",
  "Direction": "outbound-api",
  "DisconnectedBy": "Callee",
  "EndTime": "2026-05-15 16:46:50",
  "EventTime": "2026-05-15 16:46:50",
  "EventType": "terminal",
  "From": "08203612779",
  "Legs": [
    {
      "OnCallDuration": "26",
      "RingingDuration": "5",
      "Status": "completed",
      "AnsweredBy": "NA",
      "AmdCause": ""
    },
    {
      "OnCallDuration": "13",
      "RingingDuration": "4",
      "Status": "completed",
      "AnsweredBy": "HUMAN",
      "AmdCause": "HUMAN-080-880"
    }
  ],
  "PhoneNumberSid": "8224778996",
  "RecordingUrl": "https://recordings.us3.qaexotel.com/exotelrecordings/ree5e1us3/f3dec10279a310...",
  "StartTime": "2026-05-15 16:46:13",
  "Status": "completed",
  "To": "08969916092"
}
```


In this example:

- Legs[0].AnsweredBy: NA → AMD was not applicable for the From leg (Leg 1).
- Legs[1].AnsweredBy: HUMAN → the To number (Leg 2) was answered by a human.

**B. Passthru payload**

When AMD is enabled, the Passthru payload also includes AnsweredBy and AmdCause inside the Legs object.

**Sample Passthru payload:**


```
{
  "CallSid": "afc75285632df2acd33b7ea1fecb1a4h",
  "CallFrom": "06203612779",
  "CallTo": "02247789996",
  "Direction": "outbound-dial",
  "DialCallDuration": 27,
  "DialCallStatus": "completed",
  "StartTime": "2026-04-17 19:02:06",
  "CurrentTime": "2026-04-17 19:02:38",
  "Legs": [
    {
      "Number": "06355212721",
      "Type": "single",
```

```
      "OnCallDuration": 18,
      "CauseCode": "NORMAL_CLEARING",
      "AnsweredBy": "HUMAN",
      "AmdCause": "HUMAN-800-800"
    }
  ]
}
```

In this example:

- AnsweredBy: HUMAN → the call was detected as answered by a human.
- AmdCause: HUMAN-800-800 → a short greeting was detected, followed by silence – typical human behavior, where the person says "Hello?" and waits for a response.

## AMD Cause reference

Use AnsweredBy for your final classification, and AmdCause to understand why that classification was made. Values shown in {curly braces} are the actual measured/threshold figures used during detection.

| Answered By | AmdCause | What it means |
|---|---|---|
| MACHINE | INITIALSILENCE-{detected}-{threshold} | Long silence before any voice started. Machines often have a delay or beep before the greeting plays. |
| MACHINE | MAXWORDS-{wordsDetected}-{maxAllowed} | Too many words were spoken in the greeting. Machines usually have longer greetings. |
| MACHINE | LONGGREETING-{voiceDuration}-{greetingLimit} | The total voice duration of the greeting was too long. |
| MACHINE | MAXWORDLENGTH-{consecutiveVoiceDuration} | A single continuous stretch of voice without a pause was too long. |
| HUMAN | HUMAN-{silenceDuration}-{afterGreetingThreshold} | A short greeting was spoken, followed by silence. |
| NOTSURE | TOOLONG-{totalTime} | The analysis window ended before a definitive result could be detected. |
| NOTSURE | NOAUDIODATA-{totalTime} | No audio frames were received during the analysis window. |
| | | |

## Accuracy

Exotel detects voicemail using heuristic algorithms based on maximum words spoken, silence, and greeting length. Detection typically takes **3–5 seconds**, with an accuracy of **60–75%**, which is on par with industry standards.

There is no consistent signaling difference between a call picked up by a human and one picked up by a machine, so detection relies on audio cues such as tone of voice and speed of words spoken.

## FAQ

**Q: A call went to voicemail. Will its status change to "no-answer," or stay in the success bucket?**

It stays in the success bucket. The call was successfully connected and answered (by a machine), and these are billable seconds, so the call is not moved to a non-successful state. Use the AnsweredBy parameter separately to determine whether the answer was a human or a machine.

**Q: Will the AMD result appear in my reports or dashboard?**

No. AMD results are available only through the API callbacks (StatusCallback terminal event and Passthru payload), not in reports or the dashboard.

**Q: Which leg does AMD apply to?**

Only Leg 2 (the To leg). Leg 1 (the From leg) always returns AnsweredBy: NA.

**Q: Is AMD real-time? Will it delay my call?**

No delay. AMD is asynchronous – the call connects and proceeds normally while detection runs in the background. The result is delivered through the callback once detection completes (roughly 3–5 seconds).

# 8. FAQs

# 8.1. Promotional Calls to Jammu & Kashmir Numbers

You cannot place promotional or bulk calls to phone numbers that belong to the Jammu & Kashmir (J&K) telecom circle. As a safety and security precaution, Exotel blocks these call types to J&K numbers. Individual transactional calls to J&K numbers are still allowed.

## Why this restriction applies

As a safety and security precaution, Exotel does not allow bulk calls and promotional calls to numbers that belong to the Jammu & Kashmir telecom circle.

## What you can still do

You can still make individual transactional calls to J&K numbers. Only bulk and promotional calls to these numbers are restricted.

## Learn more

To understand more about this, refer to this link: **http://www.jktpo.in/contact.html**

## Need help?

If you have any questions or concerns, connect with us using the **chat widget** on your Exotel Dashboard, or WhatsApp us on **08088919888**.