exotel
DOCS

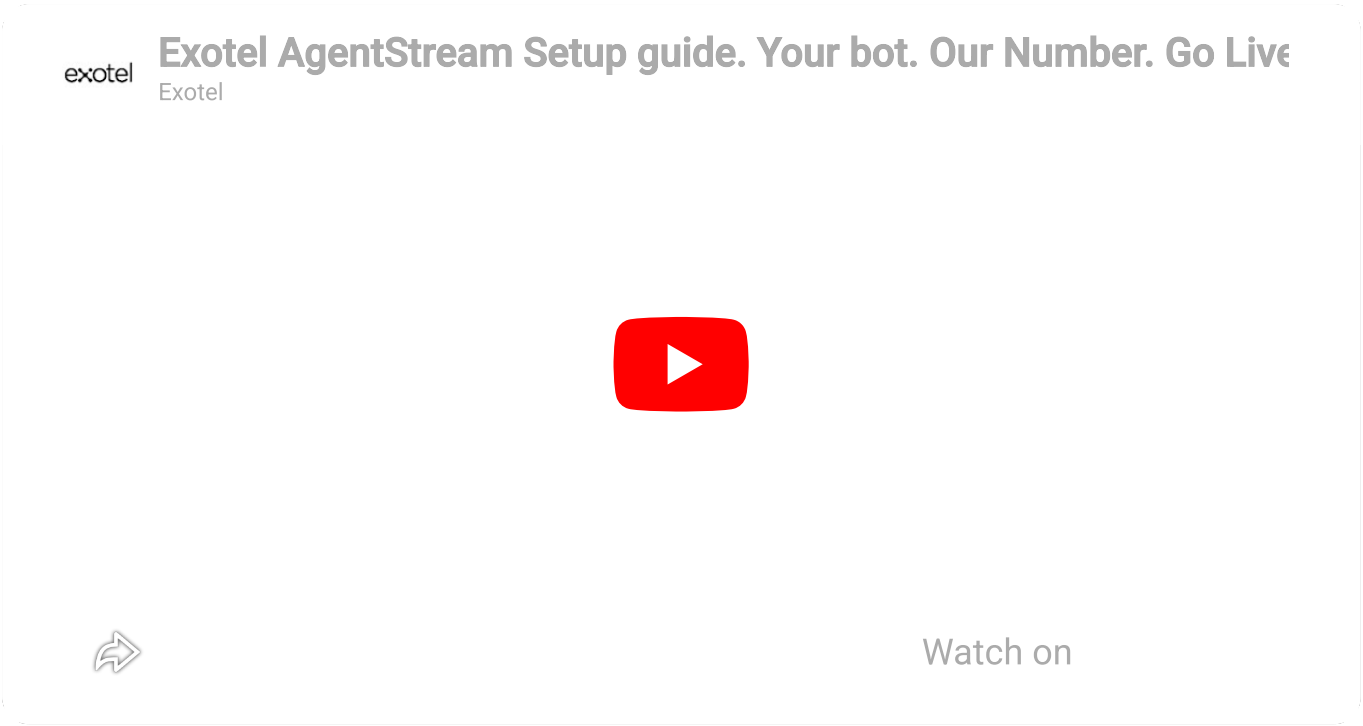
Exotel AgentStream

1. AgentStream

1.1. Get Started with Exotel AgentStream

This guide outlines how to enable and use Exotel's AgentStream capabilities via the Voicebot Applet or Stream Applet. These applets allow real-time audio streaming between your customer calls and bot platforms over WebSocket.

1.1.1. Overview & Quickstart



Step 1: Account Setup

Before enabling streaming services, ensure you have an active Exotel account in the appropriate region.

Choose the Right Instance

Instance Location	URL	Use Case
Mumbai (Veeno)	Link	Required for BFSI accounts or compliance, WebSocket + SIP Trunking, or IP-PSTN intermix scenarios(Recommended for Indian Customers)
Singapore	Link	Default region for most use cases

Note: IP-PSTN intermixing is supported only on the Mumbai (Veeno) instance.

Step 2: Complete KYC

AgentStream access requires completion of your KYC verification.

- Submit your company documentation as per standard onboarding.
- Refer to the complete KYC process here: How to complete KYC for Exotel
- Wait for confirmation from the Exotel compliance team before proceeding.

Note: KYC and streaming enablement can happen in parallel.

Step 3: Integration and Support

Once access is enabled, proceed with integration using the documentation below.

1.1.2. 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.

1.1.3. 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.

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?

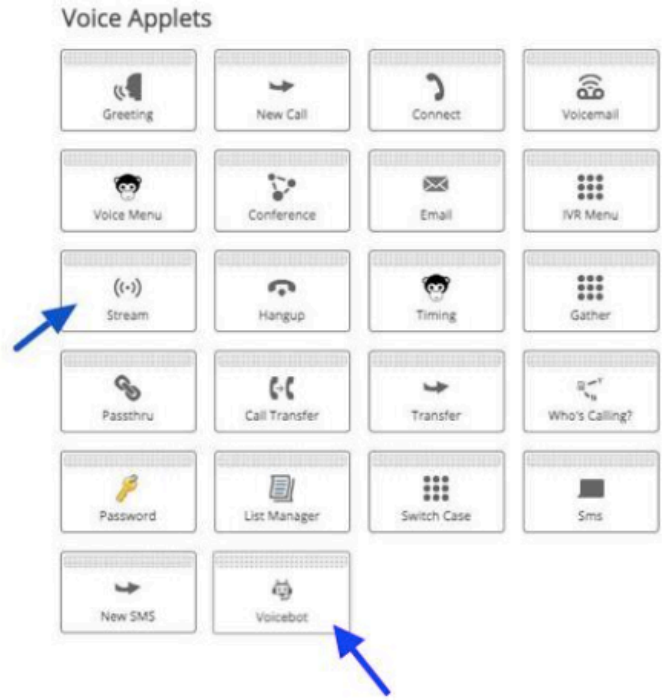
- Learn how to build flows with the Exotel Flow Builder guide.
- Configure your bot in a flow using the Voicebot Applet guide.
- For a direct bot connection without a flow, see the Connect to Bot API.
- Contact support at hello@exotel.com.

1.2. AgentStream Applets

1.2.1. AgentStream Applets

AgentStream Applets are building blocks inside Exotel that let you decide *how* audio from a live call is streamed to your bot or application over wss. Each applet controls the **direction of audio flow, event handling, and bot playback behavior**

Enabling And Disabling Streams: Unidirectional and Bidirectional Streams can be enabled for a call flow using the “Stream” and “Voicebot” Applet, respectively when creating Custom Apps in the App Bazaar.



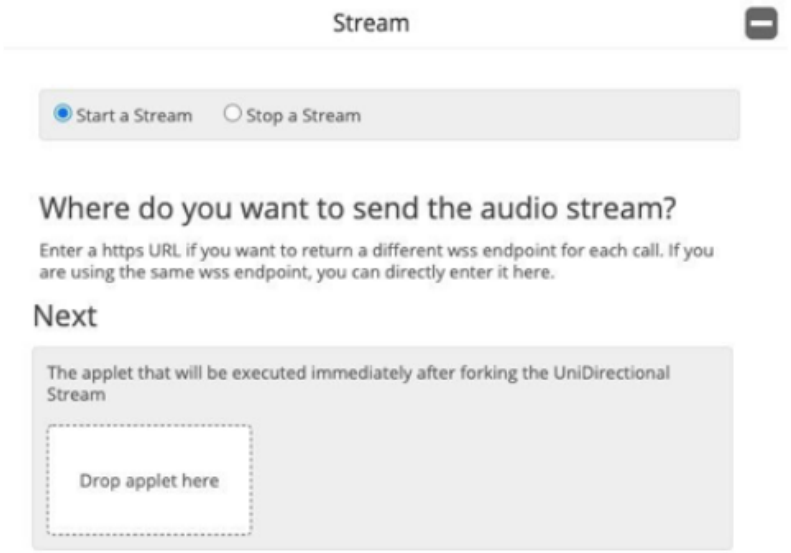
1.2.1.1. Unidirectional Streaming

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.

1.2.1.2. Stream Applet

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”

```
{
```

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

```
}
```

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

3. Next Applet

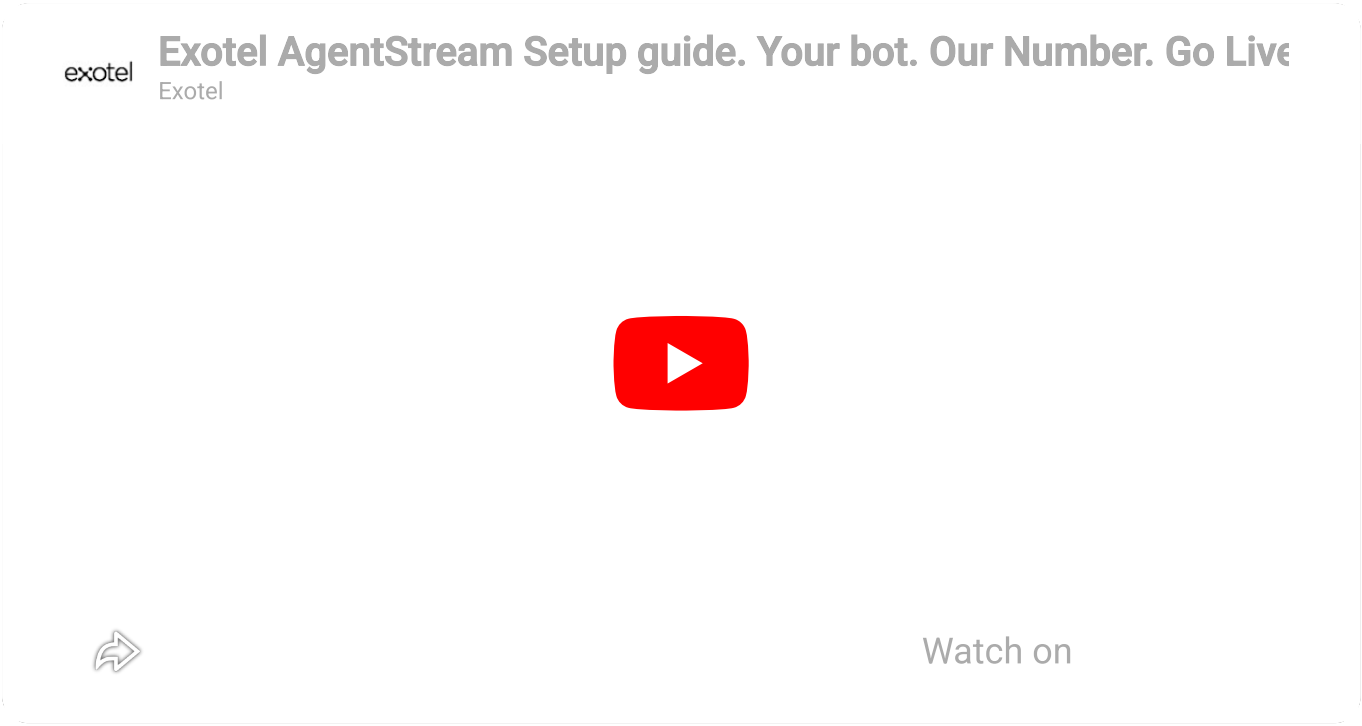
In the case if Unidirectional Stream, the Stream would be created and the call flow proceeds to the next applet configured

1.2.1.3. Bidirectional Streaming

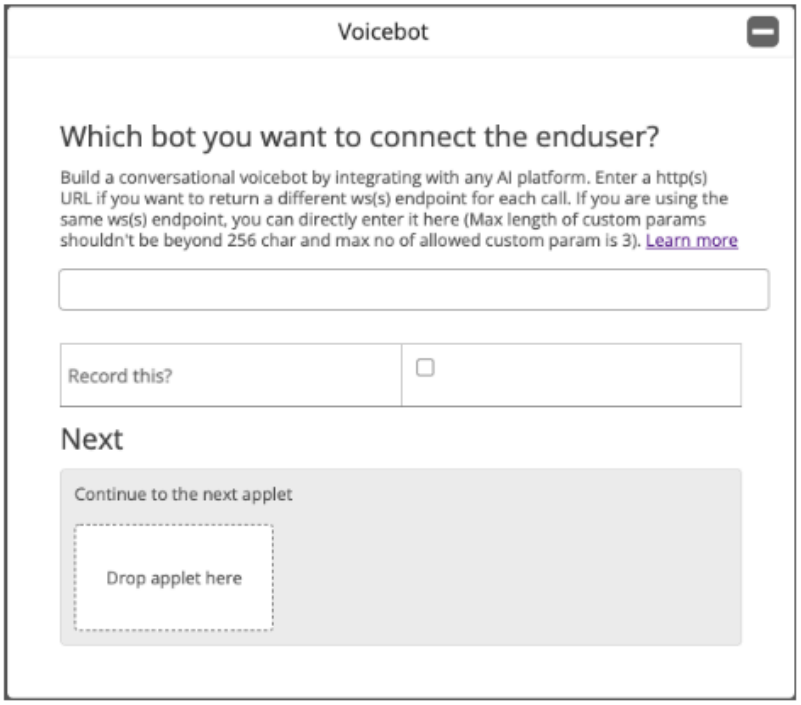
Bidirectional Streams allow two-way flow of voice data over a websocket. Exotel would send the voice data of the caller to a websocket endpoint. The endpoint can return back voice data back on the websocket and Exotel would play it out to the caller. The primary use case for this is to enable building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

1.2.1.4. VoiceBot Applet

Configuring a Bidirectional Stream- Voicebot Applet



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.

```
{"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.

```
{
"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.

```
{
"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.

```
{
"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.

```
{
"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.

```
{"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.

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

Media message:

This message encapsulates the audio packets.

```
{
"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.

1.2.2. Passthru Applet

Overview

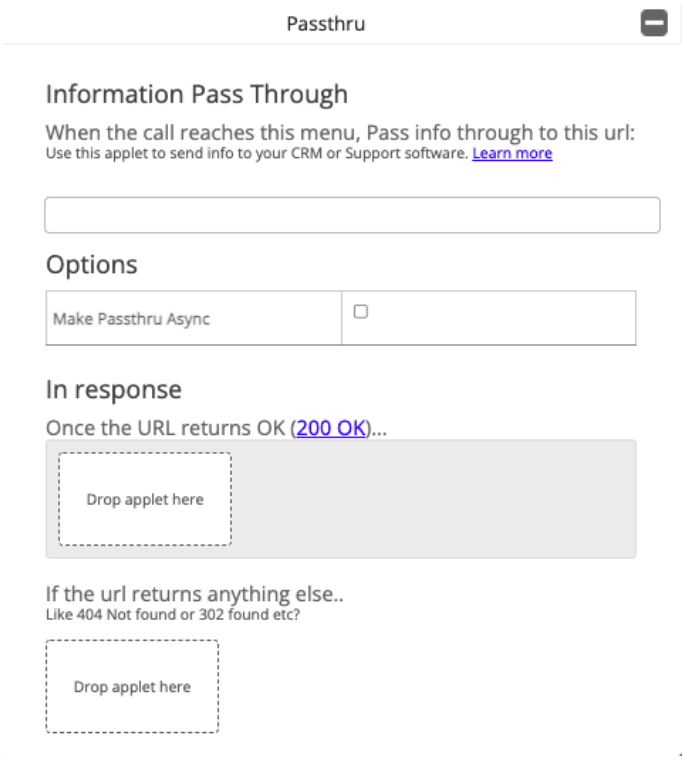
The Passthru Applet sends call and streaming metadata from Voice Applet flows to your server, especially useful when executing:

- **Voicebot Applet (Bidirectional Streaming)**
- **Stream Applet (Unidirectional Streaming)**

Passthru sends additional stream-specific parameters as query parameters, enriching standard passthru behaviour.

Refer to the standard passthru documentation here: Working with Passthru Applet

This guide is an **extended version** with additional parameters for streaming flows.



How It Works

Passthru makes an HTTP GET request to your URL, passing URL-encoded call and stream metadata.

Example:

```
GET https://yourserver.com/passthru-handler?
```

```
CallSid=56b1234567abcdef89abcdef12345678&
```

```
flow_id=voicebot-flow-xyz123&
```

```
Direction=inbound&
```

```
From=+918888000000&
```

```
To=+912233344455&
```

```
Stream[StreamSID]=6f048d2e897a0d2d4029560f3f541947&
```

```
Stream[Status]=completed&
```

```
Stream[Duration]=28&
```

```
Stream[RecordingUrl]=https://recordings.exotel.com/exotelrecordings/exotelt/6f048d2e897a0d2d4029560f3f541947.mp3&
```

```
Stream[StreamUrl]=wss://stream.customer.com/media&
```

```
Stream[DisconnectedBy]=bot
```

Your backend must parse:

- URL query parameters
- Nested keys (e.g., Stream[Status])
- JSON strings

JSON Format (Raw Query String)

Sometimes, all parameters are sent in a single JSON string under the Stream key:

```
Stream={"StreamSID":"62xxx...",
```

```
"RecordingUrl":"https://recordings.exotel.com/...",
```

```
"Status":"completed",
```

```
"Duration":"28",
```

```
"StreamUrl":"ws://10.32.76.120:5007/media",
```

```
"DisconnectedBy":"user"}
```

- Ensure your backend deserialises this string correctly.

Handling Passthru Responses

Passthru requests can be handled synchronously or asynchronously:

- **Sync**: Exotel will immediately pass the call details synchronously during the call flow execution. It is possible only to make a binary decision using Passthru.
  - `200 OK → Option A`
  - `302 Found → Option B`
  - The person on call will wait until your URL responds.
- **Async**: Exotel will asynchronously pass the call details without interrupting the call flow execution. Use this mode when you don't want to dynamically change flow execution. The user will not experience a delay.

Streaming-Specific Fields (Flat Format)

Parameter	Description
Stream[StreamSID]	Unique streaming session ID
Stream[Status]	Status (completed, failed, cancelled)
Stream[Duration]	Stream duration (seconds)
Stream[StreamUrl]	WebSocket URL for streaming
Stream[RecordingUrl]	URL to recording (if enabled)
Stream[DisconnectedBy]	Disconnector (user, bot, NA)
Stream[DetailedStatus]	Present on failure (e.g., Streaming_call_throttled)
Stream[Error]	Error string (e.g., network failure or WS rejection)

Note: The above fields are added on top of standard passthru parameters.

Throttling Scenario

Concurrency limit breaches trigger:

```
Stream[Status]=failed
```

```
Stream[DetailedStatus]=Streaming_call_throttled
```

Implement retry or fallback strategies accordingly.

Failure Handling

Below are the most common failure and disconnection scenarios captured in the passthru payload:

Test 1: Call hung up during greeting

Expected: Capture who disconnected and status

```
{
```

```
"Status": "cancelled",
```

```
"DisconnectedBy": "user"
```

```
}
```

Test 2: Invalid WebSocket URL

Expected: Failure metadata including error, stream URL, and disconnector

```
{
```

```
"StreamSID": "b68e46bca0fdexxxxx89643f6d81196d",
```

```
"RecordingUrl":
"https://recordings.exotel.com/exotelrecordings/exotelt/b68e46bca0fdexxxxx896
43f6d81196d.mp3",
```

```
"Status": "failed",
```

```
"Duration": "0",
```

```
"StreamUrl": "ws://xx.x.xx.xxx:50/media",
```

```
"Error": "3009 failed to establish ws conn dial tcp 10.1.76.120:50: connect:
connection refused",
```

```
"DisconnectedBy": "NA"
```

```
}
```

Test 3: Call cancelled after delay in streaming setup

Expected: Accurate cancellation reason and disconnection origin

```
{
```

```
"StreamSID": "6f048d2e897a0dxxxxx9560f3f541947",
```

```
"Status": "cancelled",
```

```
"DisconnectedBy": "user"
```

```
}
```


Test 4: Call disconnected by bot after successful conversation

Expected: Final stream status and session metadata


```
{
```

```
"StreamSID": "f5dd49d7ac3xxxxx4b491ce948501947",
```

```
"Status": "completed",
```

```
"Duration": "29",
```

```
"StreamUrl": "wss://e33e-54-251-51-3.xxxxx-free.app",
```

```
"DisconnectedBy": "bot"
```

```
}
```


These structured outputs ensure consistent understanding of streaming lifecycle and failure context.

Note: DisconnectedBy values are standardized to: user, bot, NA.

Passthru logs enriched with detailed Error messages improve debugging and recovery mechanisms.

1.2.2.1. More on Passthru Applet

Best Practices

- Position Passthru immediately after Stream or Voicebot Applets.
- Backend should handle both flat and JSON query formats.
- Trigger fallback based on Stream[DetailedStatus].
- Actively monitor concurrency limits.
- Log StreamSID, RecordingUrl, and detailed errors for debugging and auditing.
- Clearly interpret and handle disconnect causes (DisconnectedBy).
- Passthru backend responses must comply with synchronous (HTTP codes) or asynchronous processing models appropriately.

Use Cases

- Real-time monitoring of stream health
- Dynamic routing based on real-time stream conditions
- Enhanced debugging through detailed error reporting
- Optimised concurrency and call management
- Graceful error handling and escalation mechanisms

Deployment Strategy

Follow structured rollout:

1. **Dev Environment**: Validate passthru logic with mock logs.
2. **QA/Staging**: Test real call scenarios thoroughly.
3. **Canary Release (Prod)**: Roll out feature gradually, monitoring closely.
4. **Full Release**: Expand to all users upon successful canary release.

Ensure rollback capabilities via feature flags and previous stable versions.

Summary

The Passthru Applet for AgentStream significantly enhances standard passthru functionality by providing extensive stream metadata, failure reasons, and real-time observability. Leverage these capabilities to:

- Monitor and optimize streaming performance
- Facilitate precise error handling and debugging
- Enable intelligent and dynamic routing decisions

Implement these advanced passthru mechanisms to build robust, observable, and scalable voice-streaming integrations with Exotel APIs.

1.2.3. Active Stream Monitoring API

The **Active Stream Monitoring API** lets you query and track all **live AgentStream sessions** running on your Exotel account in real time.

Monitor active streams:

```
curl -X GET 'https://<your_api_key>:<your_api_token>
<subdomain>/v1/Accounts/<your_sid>/ActiveStreams'
```

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**

Sample Response:

```
{
```

```
"status": "success",
```

```
"active_streams": 12,
```

```
"max_allowed_streams": 100,
```

```
"account_sid": "Exotel"
```

```
}
```

1.2.4. Advanced

Updated Extension Guide: Working with the Stream and Voicebot Applet

Overview

This guide provides a comprehensive overview of how to use Exotel's **Stream Applet** (Unidirectional) and **Voicebot Applet** (Bidirectional) for media streaming applications. These applets enable real-time **audio streaming only** between Exotel and external bot platforms using **WebSocket-based** integration. Note: Exotel does **not** provide inbuilt STT (Speech-to-Text) or TTS (Text-to-Speech) capabilities. You must use your own bot platform to handle media decoding and AI processing.

This article is an **extension** of: Working with the Stream and Voicebot Applet

For stream metadata, logging, and observability, refer to the companion article: Here

Why Choose WebSocket-Based Integration?

WebSockets are ideal for low-latency, bi-directional audio streaming. Compared to SIP or polling mechanisms, WebSocket integration offers:

- **Real-time bidirectional media transfer**
- **Persistent low-latency connection**
- **Lightweight and scalable protocol for AI integrations**
- **Simplified firewall/NAT traversal compared to SIP RTP**

This makes it the preferred method for modern bot platforms such as Dialogflow, Azure Bot Framework, Gupshup, Yellow.ai, Haptik, Google CCAI, Amazon Lex, and in-house NLU/LLM-based bots that support real-time WebSocket-based audio streaming.

Key Technical Concepts from the Core Stream/Voicebot Applet Article

To ensure compatibility with the base setup, this section re-emphasises the following from the core guide:

- **WebSocket Endpoint URL** must be publicly reachable and must support ws:// or wss:// protocol with base64-encoded audio frames.
- **Maximum Connection Time**: Streaming can last up to 60 minutes per session (check limits based on plan).
- **Timeout Handling**: If the bot server does not respond within 10 seconds, the session will fail.
- **Connection Retry**: Exotel will attempt one automatic retry if the WebSocket handshake fails. Ensure redundancy in bot endpoints.
- **Security**: For wss:// endpoints, ensure valid TLS certificates are installed. Self-signed certs may be rejected.
- **Payload Format**: Use base64-decoded Linear PCM payloads (raw/slin 16 bit, 8kHz mono PCM) to feed your STT engines.

These constraints and protocols apply to both Stream and Voicebot applets and should be adhered to during bot deployment.

Applet Differences

Applet	Streaming Direction	Primary Use Case
Stream Applet	Unidirectional	Transcribe user audio, e.g., STT
Voicebot Applet	Bidirectional	Full voice AI interaction (bot speaks + listens)

Note: Exotel only handles the media relay. The bot must provide transcription, NLU, TTS, etc.

Events Sent Over WebSocket

Each applet emits events during the call session. The communication over the WebSocket is bi-directional, with distinct roles for Exotel and the bot:

- **Exotel to Bot:** Transmits session events (Connected, Start, Media, DTMF, Stop, Clear) and the audio stream in base64-encoded chunks.
- **Bot to Exotel:** Returns media (for Voicebot Applet only) and may trigger control markers (e.g., Mark). The bot must gracefully handle session events and terminate the WebSocket connection when the bot conversation ends. Ending the WSS connection

triggers Exotel to move to the next applet. There is no explicit Stop event sent from the bot to Exotel.

Each event type Exotel emits over the WebSocket serves a distinct purpose in orchestrating the media stream and enabling seamless bot interaction. Understanding these events allows for optimised bot design, real-time feedback loops, and intelligent call flow transitions.

Best Practices & Use Cases per Event:

- **Connected**: Confirms WebSocket handshake. Use this to initialise your bot pipeline (e.g., STT/TTS service initialization, session state allocation).
  - Best Practice: Log and correlate with call_sid for session tracing.
  - Use Case: Trigger bot intro prompt preparation.
- **Start**: Indicates that audio streaming is beginning.
  - Best Practice: Start buffering/streaming audio to the STT engine.
  - Use Case: Sync with a user-facing prompt like "How can I help you today?"
- **Media**: Continuous chunks of base64-encoded PCM audio.
  - Best Practice: Ensure your STT engine handles 100ms PCM blocks efficiently.
  - Use Case: Real-time speech transcription or voice intent detection.
- **DTMF**: Keypress detection on the caller's side.
  - Best Practice: Use to branch hybrid IVR+Bot logic without STT latency.
  - Use Case: Press 1 to speak to agent → Exotel hears DTMF → triggers escalation.
- **Mark**: Bot-sent event to indicate a logical milestone.
  - Best Practice: Use to sync analytics or inject debugging hooks.
  - Use Case: Mark when the bot completes a sub-flow (e.g., "address collected").
- **Stop**: Triggered by Exotel when the customer's leg is disconnected.
  - Use Case: To identify when the customer's leg has disconnected.
- **Clear**: Resets session context mid-call. Sent by Voicebot. Voicebot sends this event to indicate that the current conversational context should be flushed and re-initialized. This is useful in scenarios where the bot needs to start a fresh session mid-call, such as when the caller says “start over” or the previous context is corrupted.
  - Best Practice: Ensure your bot wipes session memory and re-establishes a clean state when a Clear event is received.

- Use Case: Caller says "start again" → Clear received → bot resets all entities, replays welcome prompt.
- Supported Chunk Size: Ensure the bot handles minimum 100ms audio chunks (approx. 3.2 KB base64 payload per frame) for seamless reset and session realignment during mid-call context switches.
- Best Practice: Use to reinitialise bot memory (e.g., context drops).
- Use Case: Caller says "start over" → bot requests Clear → reset the conversation.

These events should be monitored in real time and mapped to your backend decision logic and conversation orchestration flow.

1. Connected: Indicates the WebSocket connection is successfully established. (Initialize bot session, allocate STT/TTS/LLM resources)
2. Start: Voice media stream is starting. (Start decoding audio for transcription or detection)
3. Media: Base64-encoded PCM audio payload. (Feed to STT or analytics pipeline)
4. DTMF: DTMF tones detected. (Capture digit input in IVR scenarios)
5. Mark: Developer-defined markers. (Sync checkpoints in bot logic)
6. Stop: Media streaming session ends. (Escalate, save transcript, or trigger routing)
7. Clear: Reset session context. (Re-initiate bot logic mid-call)

Streaming Audio Format

- **Codec**: 16-bit Linear PCM (s16le)
- **Sample Rate**: 8000/16000/24000 Hz
- **Channels**: Mono
- **Encoding**: base64 in WebSocket frames

Dynamic URL and Custom Parameters

You can configure a dynamic WebSocket URL using placeholders and custom parameters.

Custom Parameter Rules:

- Max 3 parameters

- Total param length (after ?) must not exceed 256 characters
- Sample:

ws://127.0.0.1:5001/media?param1=value1&param2=value2&param3=value3

- Dynamic resolution from HTTP(s) applet must also return a valid ws(s) URL in this format.

Recording (Optional)

Recordings, if enabled, are returned via Passthru as a RecordingUrl.

Use Cases:

- Train STT/LLM models
- Compliance and QA reviews
- Escalation audits
- Voice sentiment labeling datasets

Routing to Agent or Contact Center After Voicebot Applet

When the **WebSocket connection is closed**—either due to the bot disconnecting after completing its interaction or a network-level termination—Exotel automatically moves to the next applet in the flow. There is **no explicit Stop event sent by the bot to Exotel**. Instead, **the bot must close the WebSocket session** once the conversation ends.

Upon disconnection, Exotel internally emits a Stop event and transitions to the next applet, typically a **Passthru Applet**. This Passthru makes a GET request to your endpoint, and based on your response (e.g., escalate=true), Exotel decides how to route the call.

Scenarios:

- **Connect Applet** → Route to Exotel agent
- **SIP Connect via vSIP Trunk** → Route to enterprise contact center
- **Hangup Applet** → Gracefully end the call

Example:

User: "Talk to human" → Bot finishes → WSS disconnects → Exotel emits Stop → Passthru GET call → Response: escalate=true → SwitchCase → vSIP Trunk over Connect applet

Passthru Integration

Always place a Passthru Applet immediately after Voicebot/Stream Applet to:

- Fetch session metadata
- Log streaming stats (SID, duration, Recording URL)
- Detect disconnects
- Read escalation flags from response

Best Practices

- Place Passthru immediately after streaming applet
- Use Clear/Mark events for context handling and observability
- Use Active Streams API for concurrency limits
- Design Passthru logic to interpret escalation/disconnect
- Follow WebSocket timeout, reconnect, and handshake guidelines strictly
- Keep custom params concise and secure
- Ensure bot sends Stop to gracefully close stream

Advanced Use Cases

- STT pipeline using OpenAI Whisper, Google STT
- Bot+LLM conversations via voice
- IVR replacement with NLP flows
- DTMF + Speech combo journeys
- Hybrid fallback to live agents via SIP trunk

Summary

Exotel's Voicebot and Stream Applets power modern voice automation by offering developer-first WebSocket-based audio streaming infrastructure. These applets act as programmable media bridges, streaming audio in real time between Exotel's telephony platform and any compliant bot platform capable of understanding linear PCM data.

By adopting this architecture, enterprises gain:

- **Low-latency streaming** that supports responsive AI-driven conversations
- **Vendor-neutral integration** with STT, TTS, LLMs, and custom NLP systems
- **Dynamic routing and escalation** via Passthru Applets and intelligent fallback logic
- **Secure, observable, and resilient media sessions** with active stream monitoring
- **Enterprise-grade call flow design** that integrates Exotel CC, SIP trunks, and external agents

This document is a production-ready extension to the Stream/Voicebot Applet Basic Guide and complements the Passthru Streaming Metadata Guide.

For deployment, ensure compliance with session lifecycle best practices, chunk size handling (100ms), parameter limits, recording strategies, and escalation routing logic through WebSocket termination events.

1.2.4.1. More

Capabilities Overview

- Real-time audio streaming over WebSocket (WSS)
- Voicebot Applet: Bidirectional audio interaction
- Stream Applet: Unidirectional audio streaming
- Passthru Applet: Stream lifecycle metadata, error reporting, and routing context
- SIP Trunk fallback and external call routing available via Passthru (Mumbai region only)
- Supports major bot platforms.

Best Practices

- Always choose the Mumbai instance for IP–PSTN or SIP hybrid integrations.
- Disconnect the WebSocket from your bot when the conversation is complete to allow the call flow to proceed to the next applet.
- Ensure media chunking follows Exotel's recommended size (typically 100ms PCM chunks with 3200 bytes of raw audio).
- Use the Passthru Applet after Voicebot to capture stream completion, errors, and call routing metadata.
- Avoid long delays in establishing WebSocket connections to prevent streaming failure or fallback.
- Use dynamic WebSocket URLs when handling multi-tenant or user-segmented bot sessions.
- Monitor StreamSID, Duration, Status, and DisconnectedBy via Passthru for complete lifecycle debugging.

Glossary

- **WSS (WebSocket Secure)**: Secure protocol used for encrypted audio streaming.
- **Voicebot Applet**: Enables bidirectional audio exchange between the bot and the caller.
- **Stream Applet**: Allows unidirectional audio capture from the caller to your bot.

- **Passthru Applet**: Captures stream metadata, errors, and acts as a decision trigger for routing.
- **StreamSID**: Unique identifier for each stream session, useful for logging and tracking.

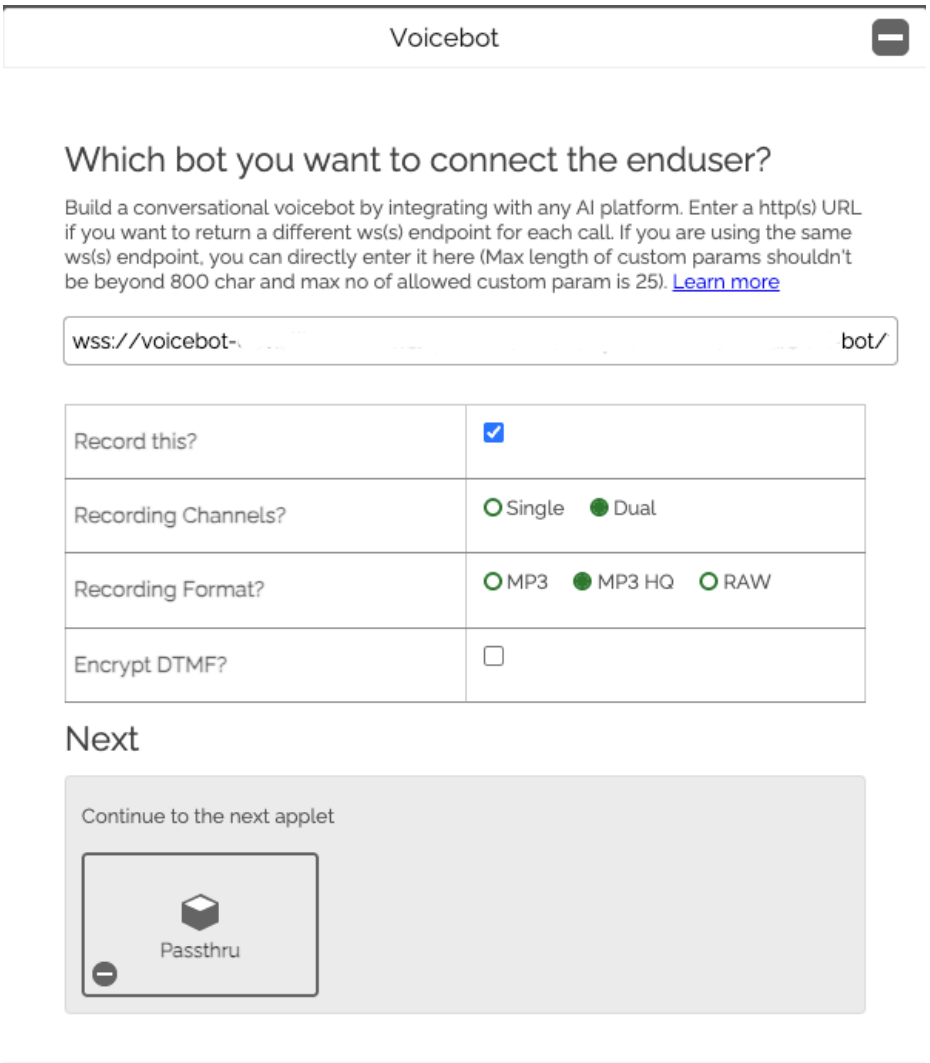
Common Use Cases

- Voicebot handling FAQ or collections for fintech and NBFCs
- Call deflection bots for first-level support in marketplaces
- Healthcare appointment management via AI bots
- Bidirectional streaming for GenAI-based interview bots
- Smart routing from bot to agent based on sentiment or keywords

Once these steps are completed and access is approved, you will be ready to stream calls to your own bot infrastructure in real time and build powerful conversational workflows using Exotel's infrastructure.

For questions, assistance, or deployment readiness, contact your CSM or raise a support request via hello@exotel.com.

1.3. Working with Recording Options in the Voicebot Applet



Voicebot Applet now allows you to record live conversations between your bot and customers for quality monitoring, compliance, or training purposes.

You can configure recording behaviour directly within the applet under the **“Record this?”** option.

Common Use Cases:

- **Compliance Audits:** Maintain proof of consent and adherence to regulatory standards.
- **Quality Monitoring:** Evaluate bot performance, tone, and speech recognition accuracy.

- **Training and Optimization:** Use call recordings to retrain AI models and improve conversation design.
- **Dispute Resolution:** Retrieve exact customer-bot interactions for support and verification.

How to Enable Recording

1. Open your **Voicebot Applet** in App Builder.
2. In the **Advanced Settings** section, enable the toggle **Record this?**
3. Once enabled, you'll see two additional configuration fields:

Recording Channels

- **Single** – Captures a mono recording (both streams mixed).
- **Dual** – Captures each participant's audio on separate channels for detailed analysis.

Recording Format

- **MP3** – Standard quality (default, 8 kbps).
- **MP3 HQ** – High-quality stereo (32 kbps/channel).
- **RAW** – Uncompressed audio for advanced analytics or machine learning models.

Note: HQ and RAW recording options are available only for accounts where the **High-Quality Recording** feature is activated. If you don't see these options, please reach out to your Account Manager or email **hello@exotel.com**.

1.4. AgentStream: WSS errors and handling

Short guide for AgentStream applets — **Stream** (unidirectional) and **Voicebot** (bidirectional) — when you host a **wss://** endpoint. Exotel **opens the WebSocket to your server**; JSON events and post-call outcomes follow the main **AgentStream** docs.

Two places errors show up

Layer	What it is	Where
WebSocket close code	Standard (RFC 6455) when the socket ends	Logs from **your WebSocket server** (the side that accepts Exotel's connection), proxies, load balancers
Streaming outcome	Success/failure + optional detail text	**Passthru** and other **callbacks** — often nested as **Stream** (same idea may appear on Status Callback depending on your setup)

They are **not** the same signal. Correlate with **CallSid** and time.

Common WebSocket close codes

Code	Meaning
1000	Normal close
1001	Endpoint going away (restart, navigate, etc.)
1002–1003	Protocol / data type problem
1006	Abnormal – often **no proper close frame** (network drop, TLS, timeout, LB idle timeout, crash). Very common; check path and timeouts first.
1007–1009	Payload / policy / size
1011	Server error while handling the connection
1012–1013	Restart / try again (depends on stack)

1005 and **1015** are reserved or synthetic in many APIs – you may still see them in logs.

Other products may use **IANA-registered application codes** (e.g. 3xxx) or **private use 4000–4999** on **your** infra – not the same as text in **Stream.Error**.

If the socket drops often: validate **wss://**, cert, **firewall / IP allowlist**, proxy **idle timeouts**, and server **cold start**. For Support: **CallSid**, **UTC time**, close **code** + **reason**, server log snippet (redact secrets).

Stream.Error in callbacks

When streaming fails, **Stream.Error** may contain a **short diagnostic string**. Treat it as **opaque** – copy the **full value** for Exotel Support; you don't need to parse it.

Interpret together with **Stream.Status**, **DisconnectedBy**, **Disposition**. If **Status** and **Error** seem inconsistent, trust the **error detail** and escalate with both.

Typical focus areas

Situation	Check
Won't connect	URL, DNS, TLS, firewall, allowlist
Times out during setup	Cold start, CPU, proxy timeouts
Dies on first events after connect	Upgrade path, auth, edge limits
Mid-call	Load, backpressure, your app latency
Voicebot-only	Bidirectional behaviour per AgentStream docs

Passthru: Stream fields (when Voicebot or Stream ran before Passthru)

Present **when the event store has them** – field names:

StreamSID, StreamUrl, Status, Duration, RecordingUrl, Error, Disposition, DisconnectedBy, DetailedStatus

Leg1RingingDuration may sit **next to** Stream, not inside it. After **Connect**, you may also get **Legs**, **DialCallStatus**, **DialWhomNumber**, etc. Some parameters are **tenant-specific** – if something is missing, check your account docs or Support.

Encoding (query vs JSON) follows your **Voice / App** integration doc.

Quick checklist

- Make Passthru handlers **idempotent** (retries, dedupe on CallSid).
- Log **CallSid**, **StreamSID**, close **code**, full **Stream.Error**.
- Return **200** quickly; don't log **credentials** in URLs.

See AgentStream Applets and the **Unidirectional / Bidirectional** pages on Exotel Docs for event shapes.

1.5. Programmable Voice APIs with AgentStream

When you build bot-driven voice experiences, reducing perceived latency is critical. With Exotel ExoML, you can control call legs and media actions in real time. This guide explains two recommended approaches:

- **Approach 1**: Create a customer leg and immediately start a stream.
- **Approach 2**: Create a leg, start a stream, and optionally play a short greeting (say/play) in parallel while the bot connects.

Both flows are supported with existing Exotel ExoML.

Prerequisites

- An Exotel account and valid account_sid, key and Token
- An exophone configured for outbound calls
- A gRPC endpoint to receive leg events.
- A WebSocket bot server that supports Exotel streaming and events

Approach 1: Create Leg + Start Stream (Bot-First)

This is the simplest flow: connect the customer leg, then hand it directly to your bot.

Step 1. Create the leg

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

Content-Type: application/json

`{`

`"contact_uri": "074xxxxx773",`

`"exophone": "0204xxxxx38",`

`"leg_event_endpoint": "grpc://<your-example-endpoint>",`

`"timeout": 30`

`}`

You will receive events such as leg_connecting, leg_ringing, and finally leg_answered.

Step 2. Start the stream

On leg_answered, start the stream:

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

Content-Type: application/json

`{`

`"direction": "bidirectional",`

`"url": "wss://bot.example.com/stream",`

`"content_type": "audio/x-mulaw;rate=8000"`

`}`

At this point, audio flows between the customer and your bot.

Approach 2: Create Leg + Start Stream + Say/Play (Parallel)

In many customer-facing scenarios, even a 1–2s delay feels like silence. To improve perception, you can play a short greeting while the stream session is warming up.

Step 1. Create the leg

Same as Approach 1.

Step 2. Start actions in parallel

On leg_answered, issue two actions:

Start stream

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

```
{
```

```
"direction": "bidirectional",
```

```
"url": "wss://bot.example.com/stream",
```

```
"content_type": "audio/x-mulaw;rate=8000"
```

```
}
```

Start greeting (choose one):

- **Say**

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

```
{
```

```
"text": "Hello",
```

```
"loop": 0
```

```
}
```

- **Play**

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

```
{
```

```
"url": "https://example.com/contact-center-tone.wav",
```

```
"loop": 0
```

```
}
```

Step 3. Stop greeting when the stream is ready

As soon as you receive the stream_started event, stop the placeholder:

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

or

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

This ensures the bot audio seamlessly takes over without overlap.

When to Use Each Approach

Scenario	Recommended Flow
Bot-first, IVR-style journeys	Approach 1
Sales / Collections (instant feel)	Approach 2
Contact center simulation	Approach 2
Tech support bot only	Approach 1

Key Notes

- **Parallelism**: Exotel supports issuing start_stream and start_say/start_play together on the same event.
- **Greeting length**: Keep placeholder audio short (200–500 ms) so the bot can take over naturally.
- **Stop timing**: Always stop say/play when you receive stream_started.

Next Steps

- Get ExoML enabled for your account
- Explore StopSay/StopPlay timing to fine-tune transitions.

1.6. StreamKit

StreamKit Developer Quickstart Guide

Bridge your Contact Centre (SIP) and Voicebot (WSS) using Exotel StreamKit Cloud Connector

1. Overview

StreamKit is Exotel's cloud-based SIP ↔ WSS connector that allows you to connect any SIP-based contact centre (like Avaya, Genesys, Ameyo, or custom PBX) with any WSS-based AI platform (Dialogflow CX, OpenAI, AWS Lex, etc.) in real time.

This document will help you set up an end-to-end integration using:

- **Exotel Dynamic SIP Trunking APIs**
- **AgentStream Applets (Stream and Voicebot / Passthru)**
- **Custom SIP header and routing configuration**
- **Voicebot and Agent Assist integration flows**

2. Prerequisites

Requirement	Description
Exotel Account	Sign up for an Exotel account by selecting browser calling and get your `Account SID, Auth Key,` and `Auth Token.`
Virtual Number	Buy or provision a virtual number/Exophone (DID) from Exotel by reaching out to Hello@exotel.com and requesting for VoIP Exophone In case, you need the IP<>PSTN Intermix request for Veeno Exophone
SIP/IVR System	A contact centre or PBX that supports SIP INVITE routing.
Voicebot Endpoint	WSS server endpoint or AI provider URL (e.g., WebSocket URL:ws://127.0.0.1:5001/media).
IP Whitelisting	Your Contact Centre PBX/Server IPs must be whitelisted on Exotel.
Exotel Singaling and Media Ips	Your Contact Centre firewall must whitelist Exotel IPs and Ports
Test	Do test calls, routing, and in case any support is required, reach out to Exotel Phone System

3. Step-by-Step Setup

Step 1: Create an Account and Virtual Number

Signup on Exotel **Link** and select browser calling while signing up

Reach out to hello@exotel.com to procure the VoIP Exophone

Once done, note:

- **API Settings Link**
  - **Account SID**
  - **Subdomain**
  - **Auth Key**
  - **Auth Token**
- **Virtual Number/Exophone Link**

Step 2: StreamKit Cloud SIP trunking Setup

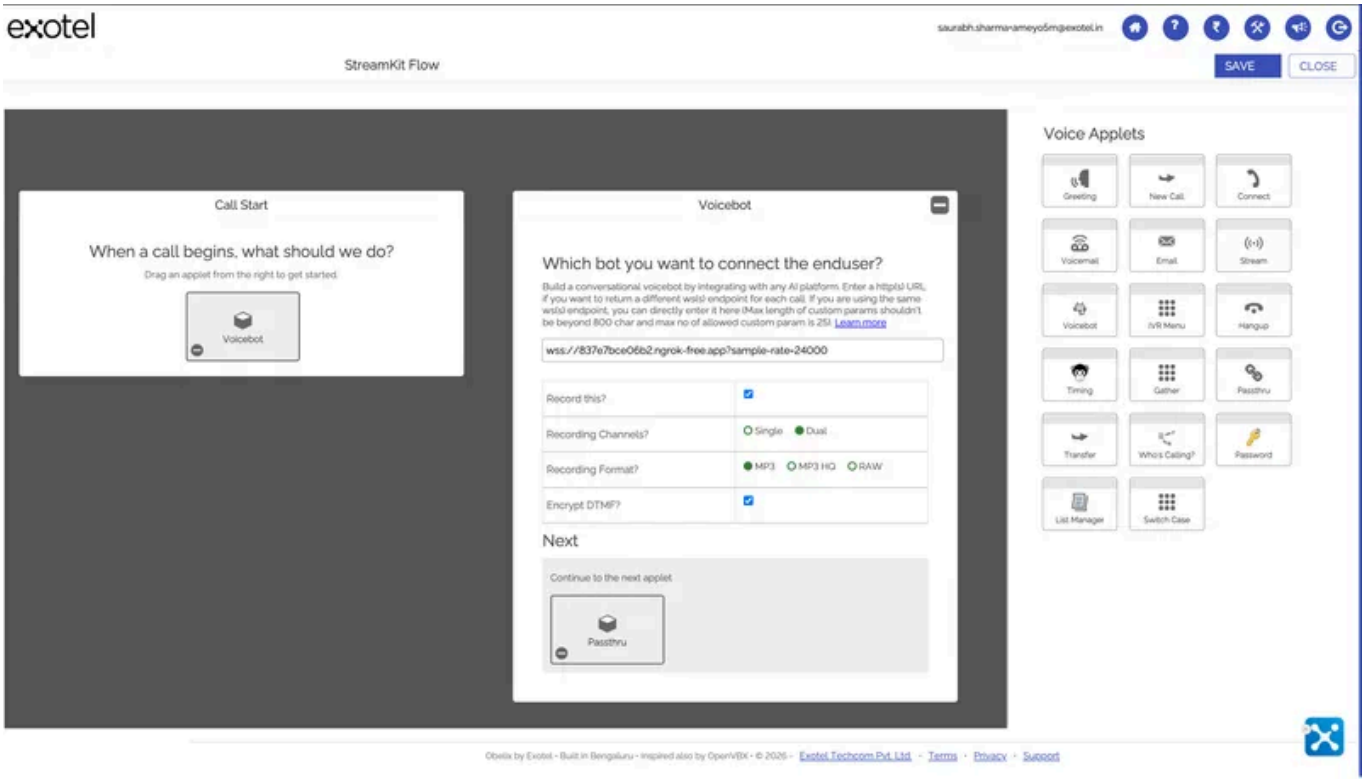
Whitelisting ensures Exotel can securely send/receive SIP traffic from your servers.

- You can add multiple IPs.
- for more details refer **Detailed SIP Trunking API Reference**

Step 3: Set Up Your Call Flow

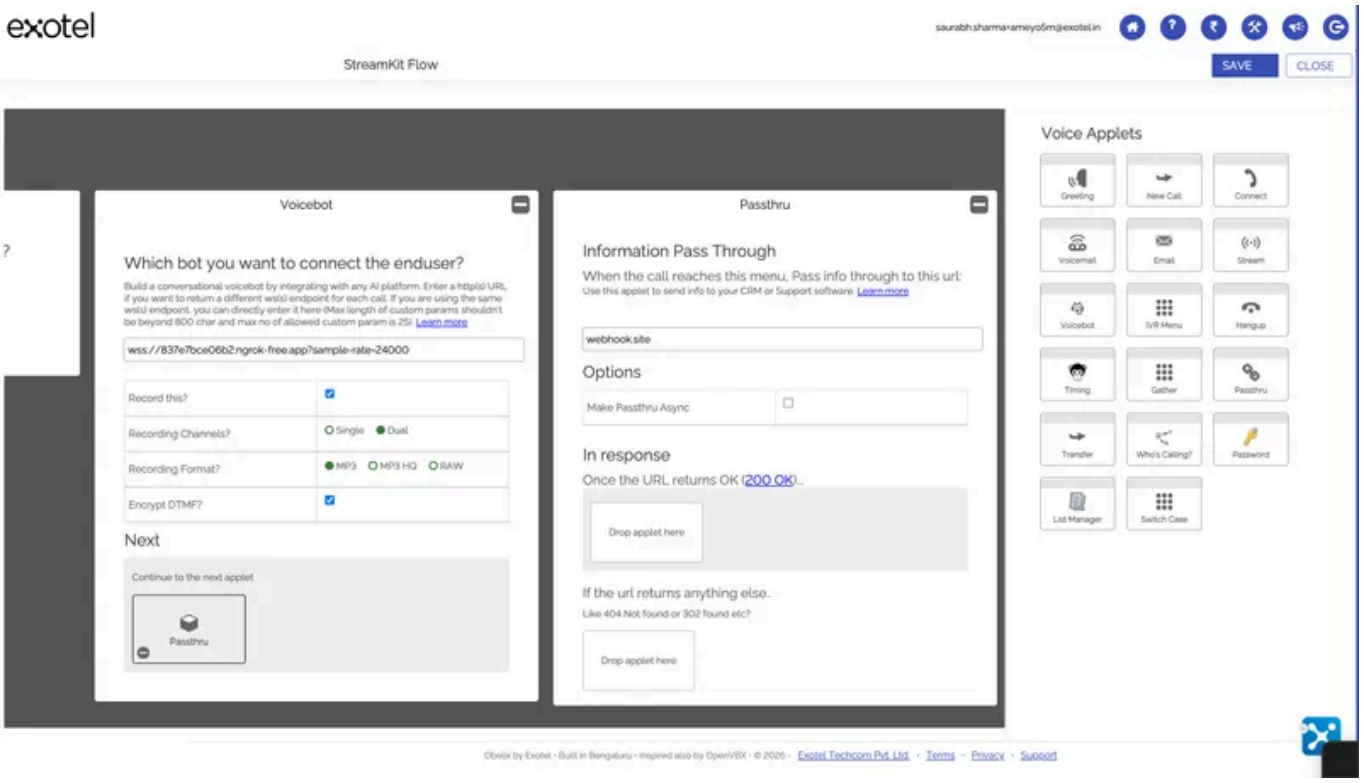
Use Exotel **Flow Builder** to define how the call moves from PSTN → SIP → WSS.

1. Log into Exotel Dashboard → **Create a Flow**
2. Add **Voicebot Applet** to connect SIP calls to your Voicebot.



Reference: **AgentStream Applets**

3. If you plan to integrate **Agent Assist**, use Stream Applet.
4. **Passthru Applet**.



Reference: **Passthru Applet**

5. **Programmable Connect: Working with Connect Applet (Dynamic URL)** in case you want to route a call back to the contact centre
6. We have to add the greeting applet as the first applet with text as ...
    - Note: if we don't add this, in that case, we will observe the voice blank issue, and the call will get disconnected in around 20 to 23 seconds
7. Map VoIP Exophone to the flow **Using Existing Phone Number** (for VoIP exophone procurement reachout to support)

Step 4: Send SIP INVITE at the Right Stage

From your Contact Centre or PBX, trigger a **SIP INVITE** to the Exotel SIP domain with required headers. Please do followthe mandatory headers as per the setup guide

Refer aswell **Network & Firewall Configuration**

Use this flow:

```
Customer → Contact Center (SIP PBX) → SIP INVITE → Exotel StreamKit → Voicebot (WSS)
```

Detailed setup guide:

Exotel SIP Trunking to Flow Integration Guide

Please do send X-Exotel-AccountSid:(Account_SID) for correct routing and CLI capture-Mandate

Step 5: Add Custom Headers

In SIP invite Contact center can pass custom headers ,

Custom headers help StreamKit voicebot applet decide which bot or campaign to route the call to.

Sample headers:

```
X-Custom-Header: SalesBot
```

```
X-UUI: XXXXXX
```

Supports upto 45 bytes

Note: Currently, additional custom headers above are supported on VoIP Exophone only.

Docs:

Flow and API Configuration Guide for Voice AI & Contact Centre Platforms

Step 6: Talk to the Voicebot (WSS)

Once the SIP INVITE is accepted:

- StreamKit opens a WSS connection to your voicebot.
- The SIP custom headers(X-UUI and X-Custom-Headers) are supported in two ways
  - It is received as part of the start event custom_parameters of the Voicebot/Stream Applet
- **custom_parameters** Key1 value is fetched from wss query params and passed at start event

```
Start Message received:
{"event":"start","stream_sid":"b6c581dXXXXXX67a2199q","sequence_number":"1","
start":
{"stream_sid":"b6c581xxxxxxxxxbfe027a2199q","call_sid":"2449xxxxx1a64fbxxxx19
```

```
9q","account_sid":"Tenant-
ID","from":"sip:0806xxxx509","to":"015xxxx0321","custom_parameters":{"Custom-
Header":"Salesbot","UUI":"XXXXXXX","key1":"value1"},"media_format":
{"encoding":"base64","sample_rate":"8000","bit_rate":"128kbps"}
```


- Received as additional SIPCustomHeader Query parameters in https `GET` request of programmable applets: Voicebot, Stream, Passthru and Connect.

Sample:

```
"SIPCustomHeader": {
    "Custom-Header": "SalesBot",
    "UUI": "XXXXXXX"
```

- RTP ↔ PCM16 media is exchanged.
- Bot starts real-time ASR/TTS or speech-to-speech conversation.

Reference Doc: **Voicebot Applet and bidirectional Integration Guide**

Step 7: Notify or Escalate via Passthru Applet

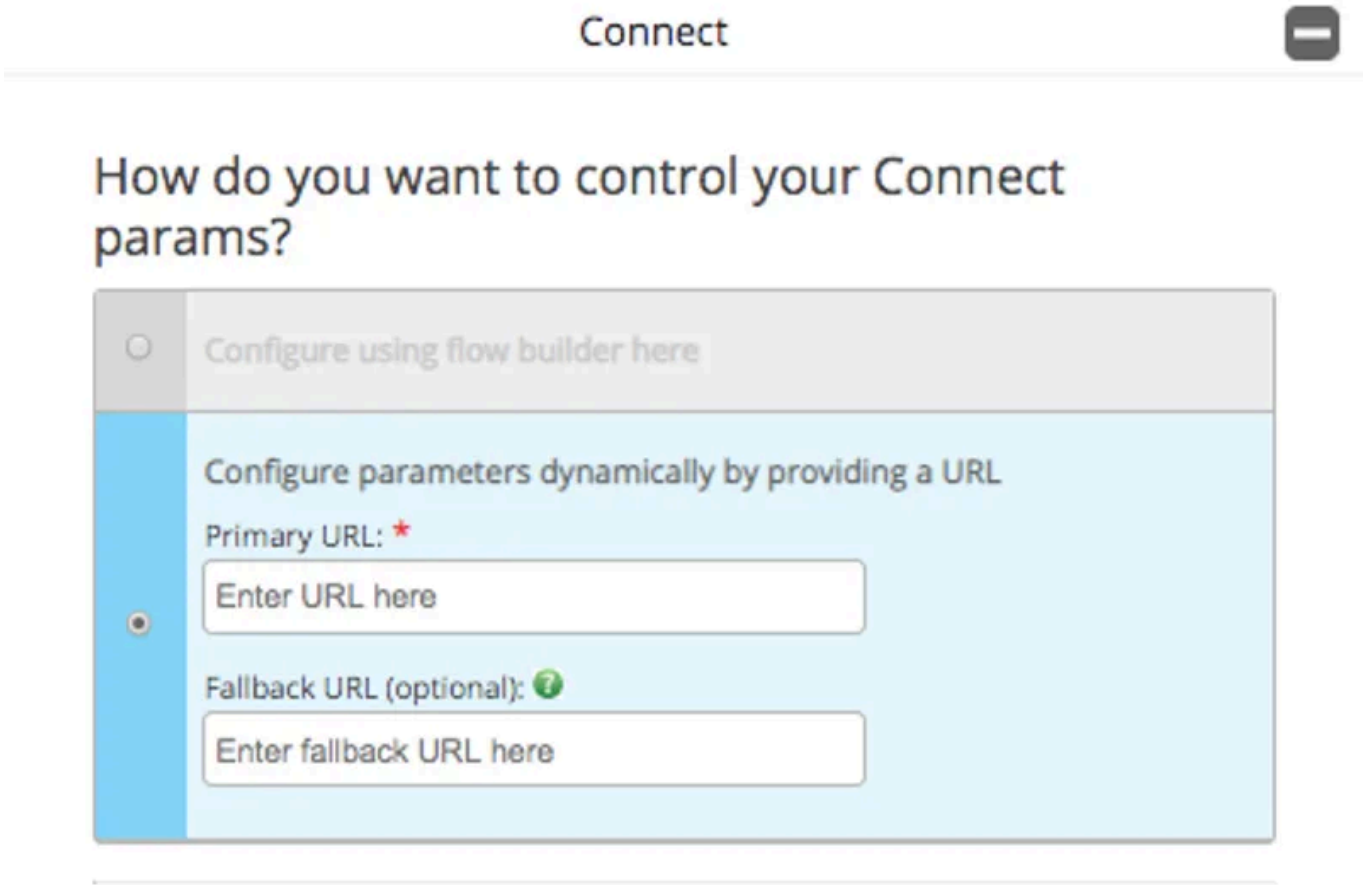
Use **Passthru Applet** to notify your contact center or escalate to an agent.

Supported modes:

1. **Notification Mode:** Bot triggers webhook via **Passthru Applet.**
2. **Transfer Mode:** Bot requests SIP Transfer to transfer call to agent by creating another trunk at Exotel

API Steps to create a Trunk

- **Detailed SIP Trunking API Reference** for Inbound call
- Create a Flow using **Connect Applet** in App Bazaar
- Use `sip:<TrunkID>` in the **Dial Whom** field
- For custom routing, use a Dynamic URL to fetch the destination URI and pass headers



- Supports **up to 3 custom SIP headers**, e.g., X-param1=value1 (max 200 bytes total)
- Headers prefixed with Exotel- or Veeno- are platform-reserved

Connect Applet – Dynamic URL Response Example:

```
{
"fetch_after_attempt": false,
"destination": { "trunk": "trunk-2134" },
"custom_params": "param1=value1&param2=value2",
"record": true,
"recording_channels": "dual"
}
```

Reference: **Programmable Connect: Working with Connect Applet (Dynamic URL)**

Docs:

Passthru Applet (AgentStream Beta)

4. Special Scenarios

Scenario	Description
Voicebot Mode	SIP ↔ WSS full-duplex. Bot handles ASR + TTS or speech-to-speech using Voicebot applet
Agent Assist Mode	Unidirectional WSS stream from caller to AI (for analysis, not speech output) using Stream start and stop applet
Agent Monitored Bot	Hybrid – bot monitors conversation and provides insights.
Post-Call Notify	StreamKit triggers the Passthru Applet after call completion for analytics or CRM updates.

5. Example Flow

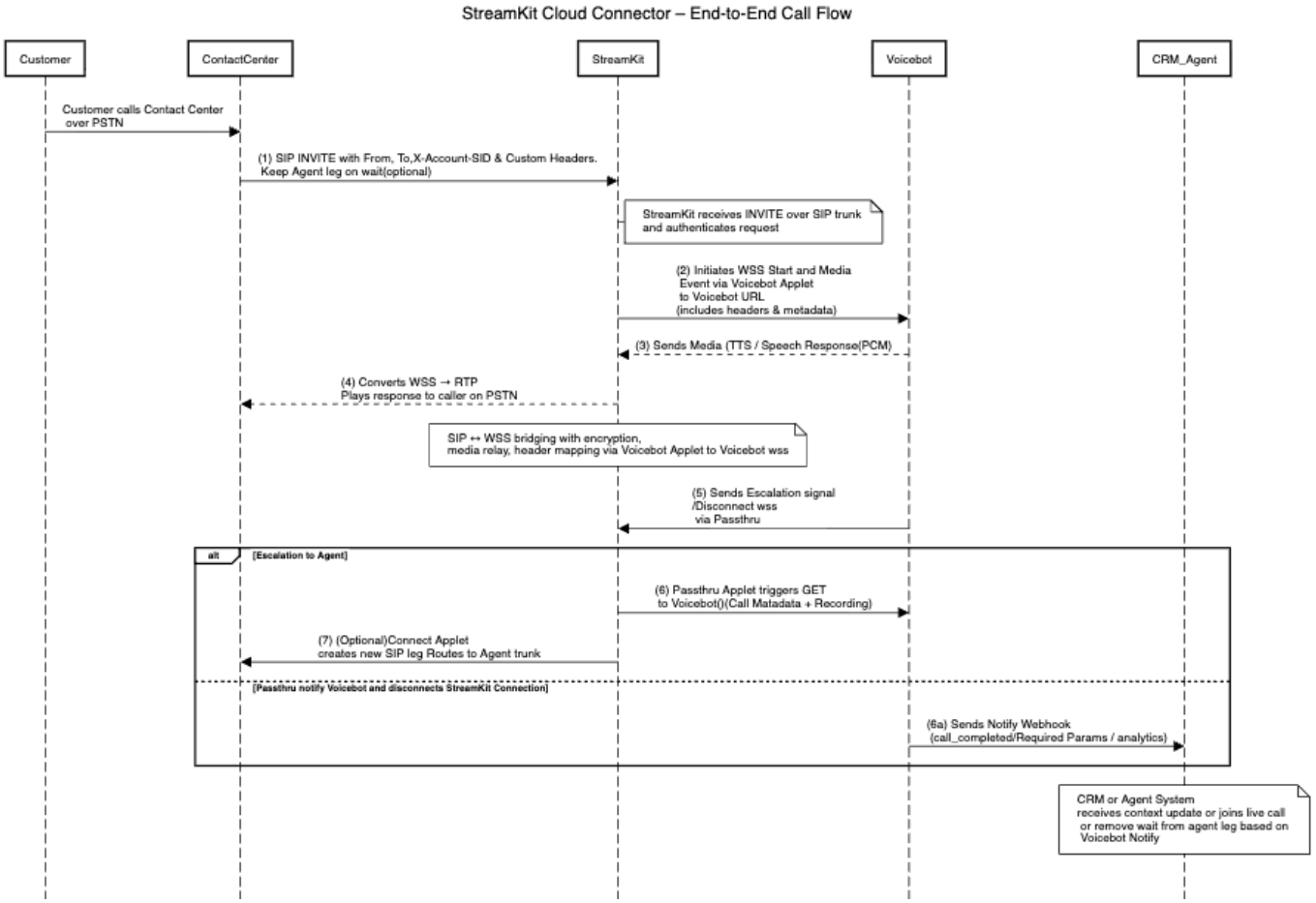
Voicebot Scenario

```
PSTN Call → CC (SIP PBX) → Exotel StreamKit → WSS Voicebot
```

Agent Assist Scenario

```
PSTN Call → CC → Exotel StreamKit (Passthru) → WSS Bot (listen only)
```

```
Bot → Notify API → CRM / Dashboard
```



6. Compliance & Data Localization

- All media and logs stay within India (Local PoPs).
- DoT and TRAI voice compliance is ready.
- Optional encrypted call recording with region-specific storage.

7. Quick Links

Resource	URL
Dynamic SIP Trunking	https://docs.exotel.com/dynamic-sip-trunking
Virtual SIP Trunk to Flow Guide	https://docs.exotel.com/dynamic-sip-trunking/exotel-virtual-sip-trunking-to-flow-integration-guide
AgentStream Applet	https://docs.exotel.com/exotel-agentstream/agentstream-applets
Passthru Applet	https://docs.exotel.com/exotel-agentstream/passthru-applet
Flow & API Config	https://docs.exotel.com/dynamic-sip-trunking/flow-and-api-configuration-guide-for-voice-ai-and-contact-centre-platforms

1.7. Connectors

1.7.1. OmniDimension Voicebot & Exotel AgentStream Integration Guide

1. Introduction

This document will help you walk through integrating Exotel phone numbers with OmniDimension (OmniDim) Voicebot, using call flows and importing into OmniDim.

What you'll achieve: incoming PSTN calls to Exotel → routed via configured call flow → handled by OmniDim VoiceBot.

2. Architecture Overview

```
Call flow:
PSTN call → Exotel → Custom call-flow (App Builder) → WebSocket / passthru
endpoints → OmniDim agent (voicebot)
```

Sequence of events with endpoints:

1. Exotel receives the call on your Exophone number
2. Exotel triggers your configured "call flow" (App)
3. Within the flow: Voicebot applet, passthru applet(s), connect/hangup logic
4. Webhook / media endpoints point to OmniDim URLs
5. OmniDim handles the voice session, agent conversation, transfer or hangup

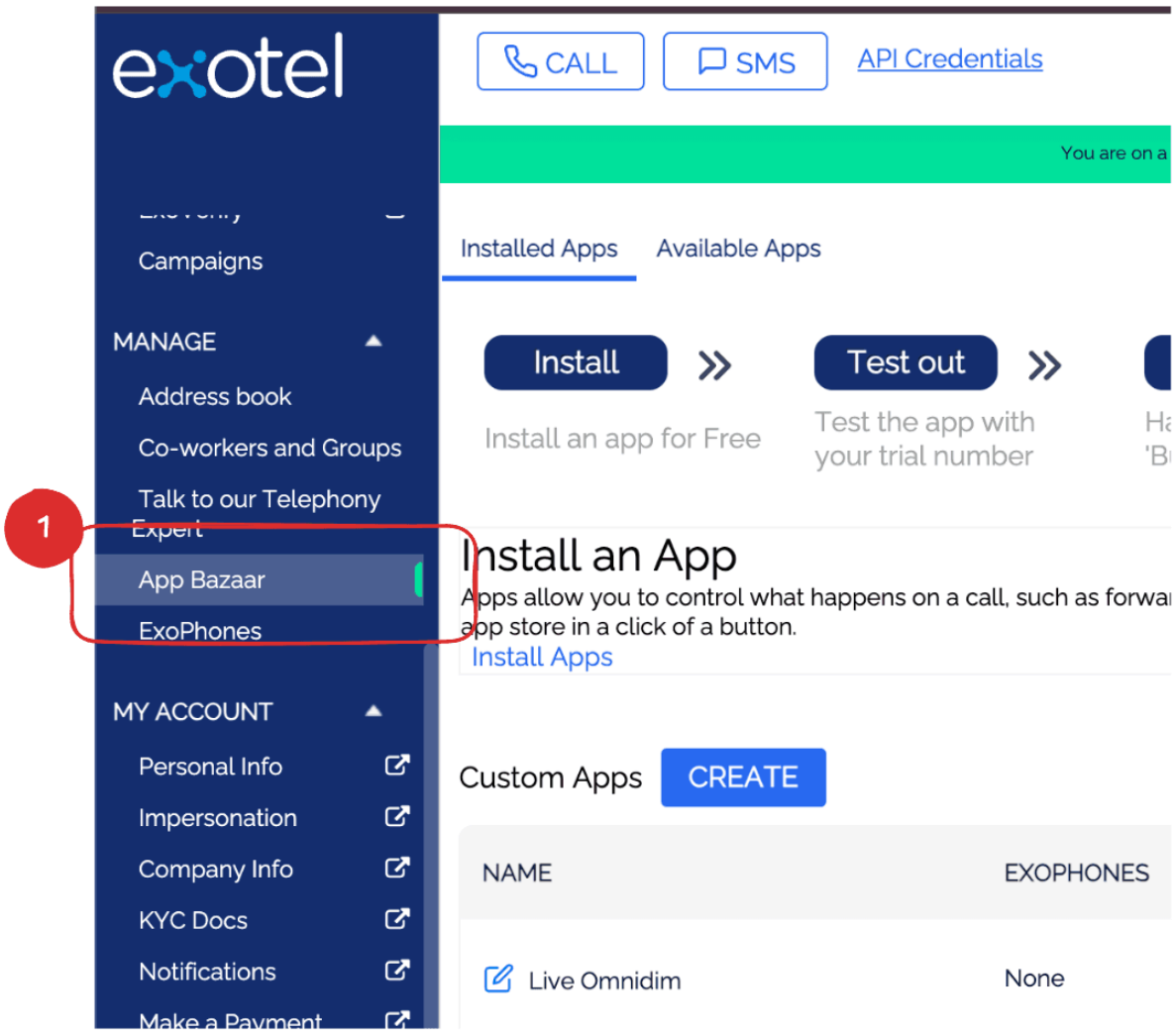
3. Prerequisites & Requirements

- Exotel account with KYC completed: **Steps to get started with Exotel**
- **An Exophone (140 series number/Landline numbers/Mobile DIDs) purchased & active on Exotel**
- **API Creds**
- Required credentials from Exotel:
  - API Key
  - API Token
  - Subdomain
  - Account SID
  - Exophone Number
  - App ID (call flow / App ID)
- Access to OmniDim portal & permission to import numbers
  URLs / endpoints in OmniDim for media/control.

4.Exotel Flow Setup

4.1 Access App Builder in Exotel

- In your Exotel dashboard, go to the side menu and click on 'App Bazaar'.
- Navigate to App Bazaar in the side menu

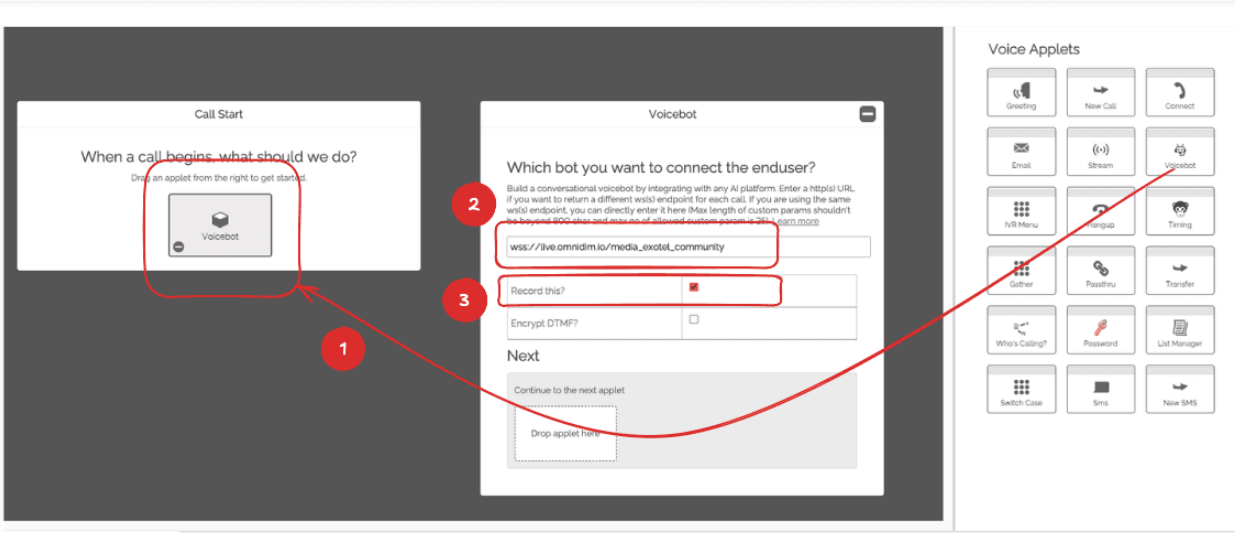


4.2 Create New App

- Click on 'Create New App' to start building your call flow.
- Click Create New App button

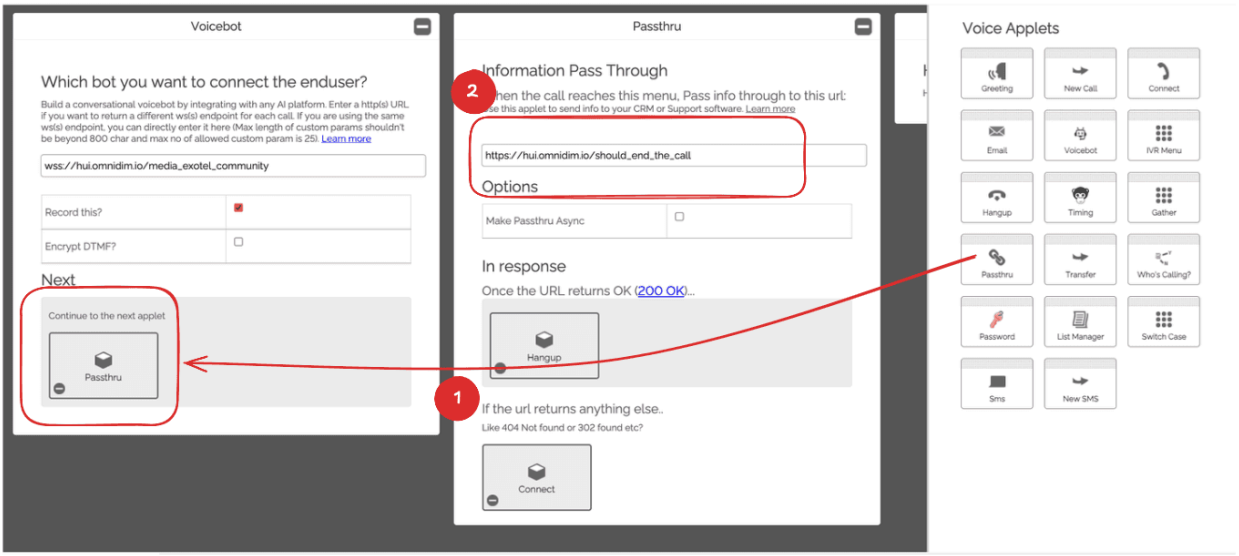
4.3 Add Voicebot Applet for Call Coming

- On the 'Call Start' block, drag a 'Voicebot' applet from the right sidebar. In the URL field, enter: wss://live.omnidim.io/media_exotel_community and also turn recording ON.
- Drag the Voicebot applet and add the WSS URL



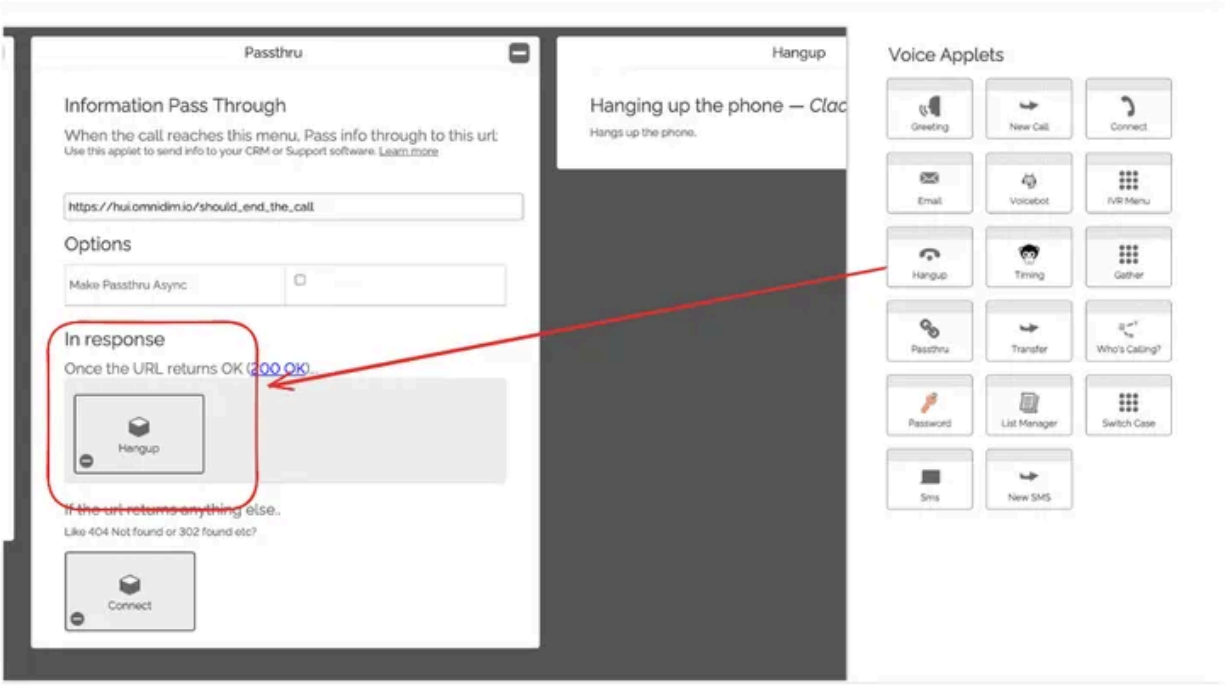
4.4 Add Second Passthru Applet

- On the voicebot applet's next section, drag another 'Passthru' applet. In the URL field, enter: https://live.omnidim.io/should_end_the_call
- Add second passthru applet



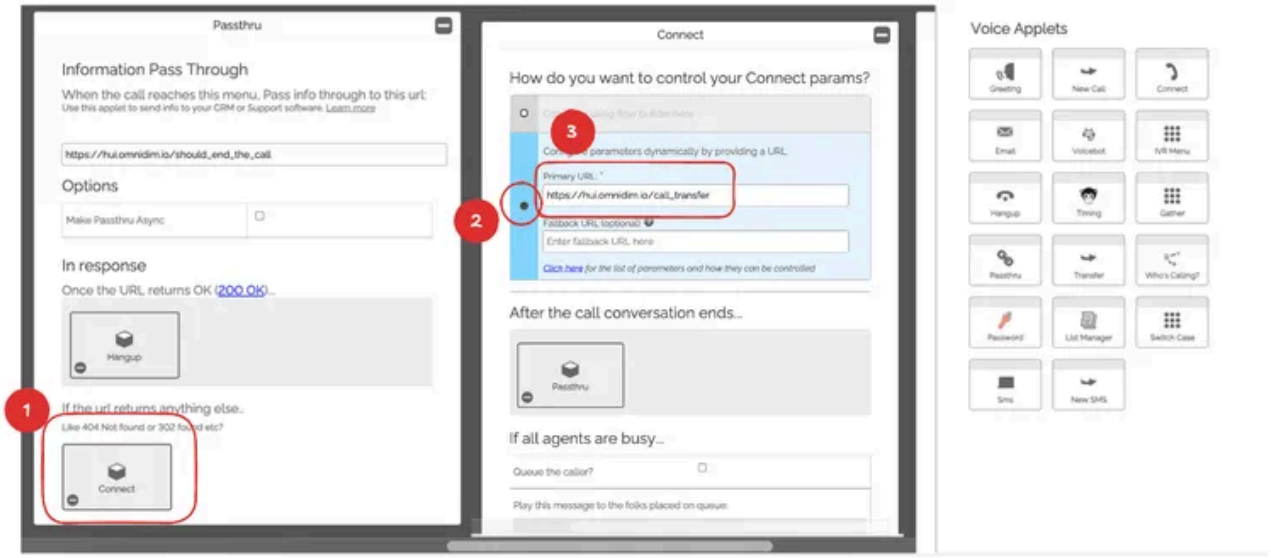
4.5 Add Hangup on 200 Response

- In the 200 response of the second passthru applet, drag a 'Hangup' applet to end the call when needed.
- Add hangup applet on 200 response



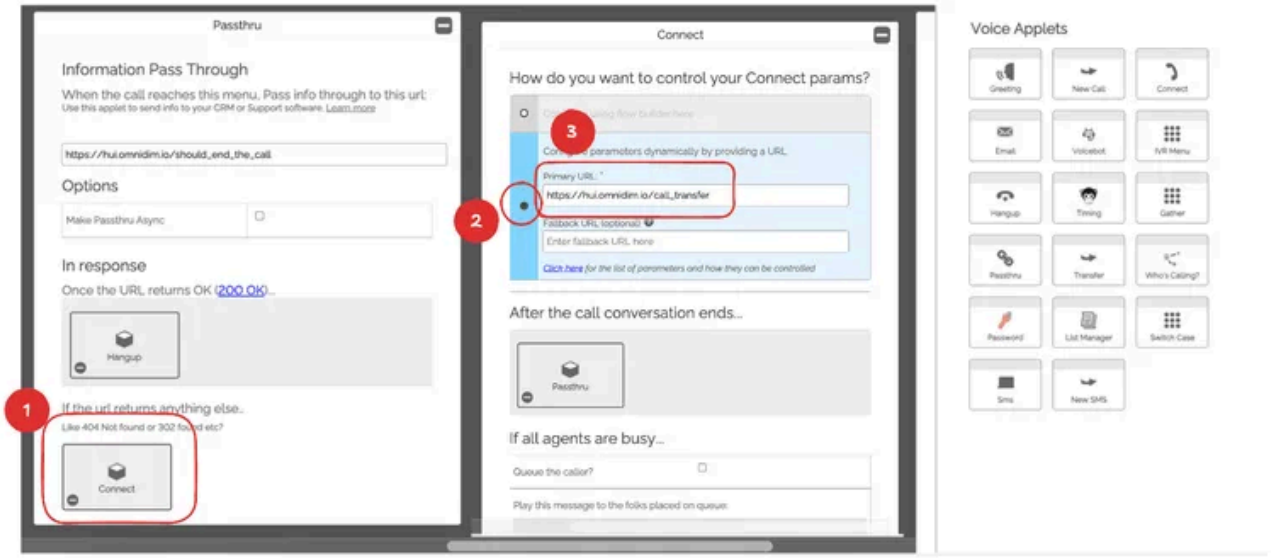
4.6 Add Connect Applet for Non 200 response

- In the 404 or 302 response of the second passthru applet, drag a 'Connect' applet for call transfers.
- Add connect applet on 404 or 302 response



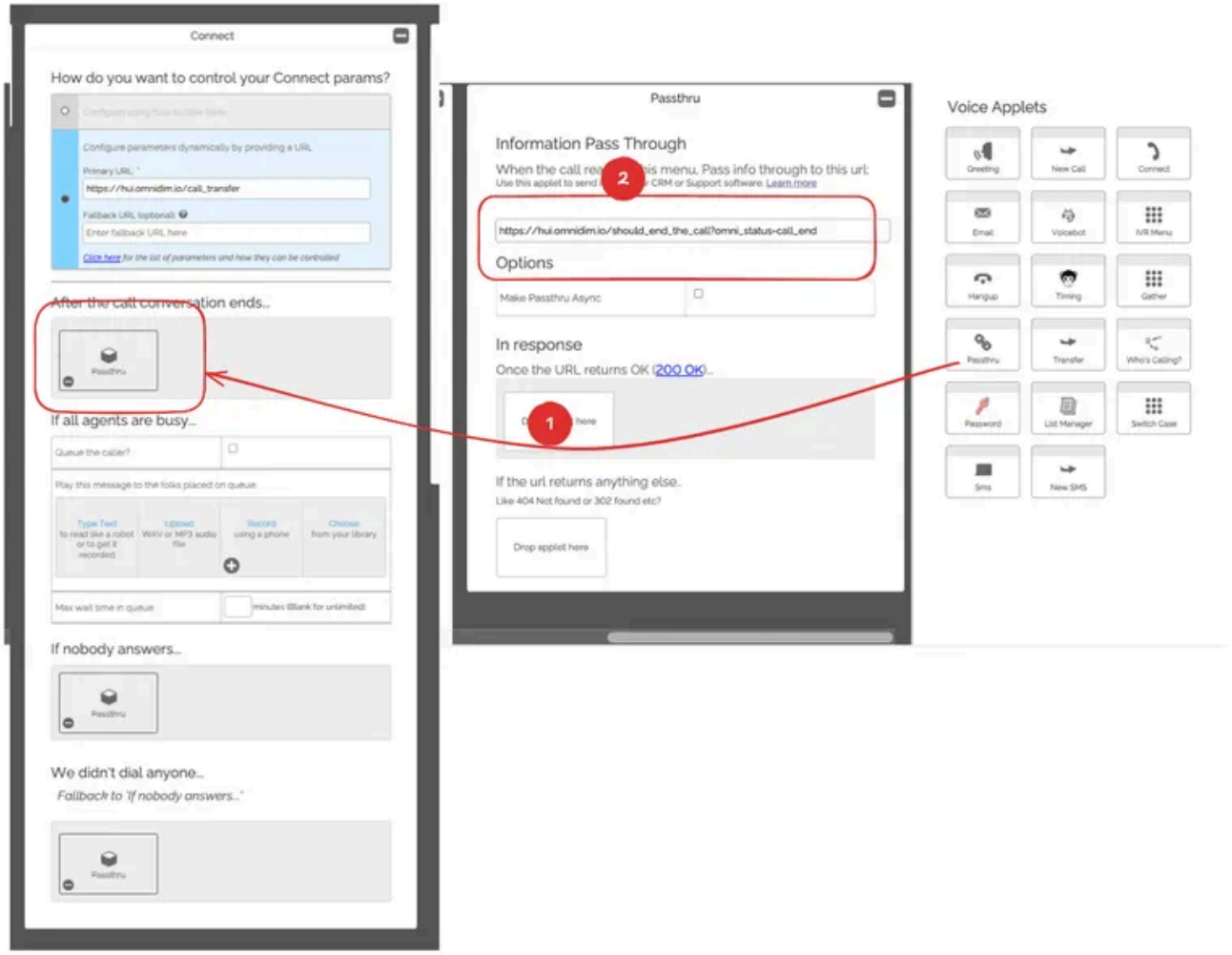
4.7 Configure Connect Applet

- In the connect applet, select 'Primary URL' and enter: https://live.omnidim.io/call_transfer
- Configure the Connect applet with the URL



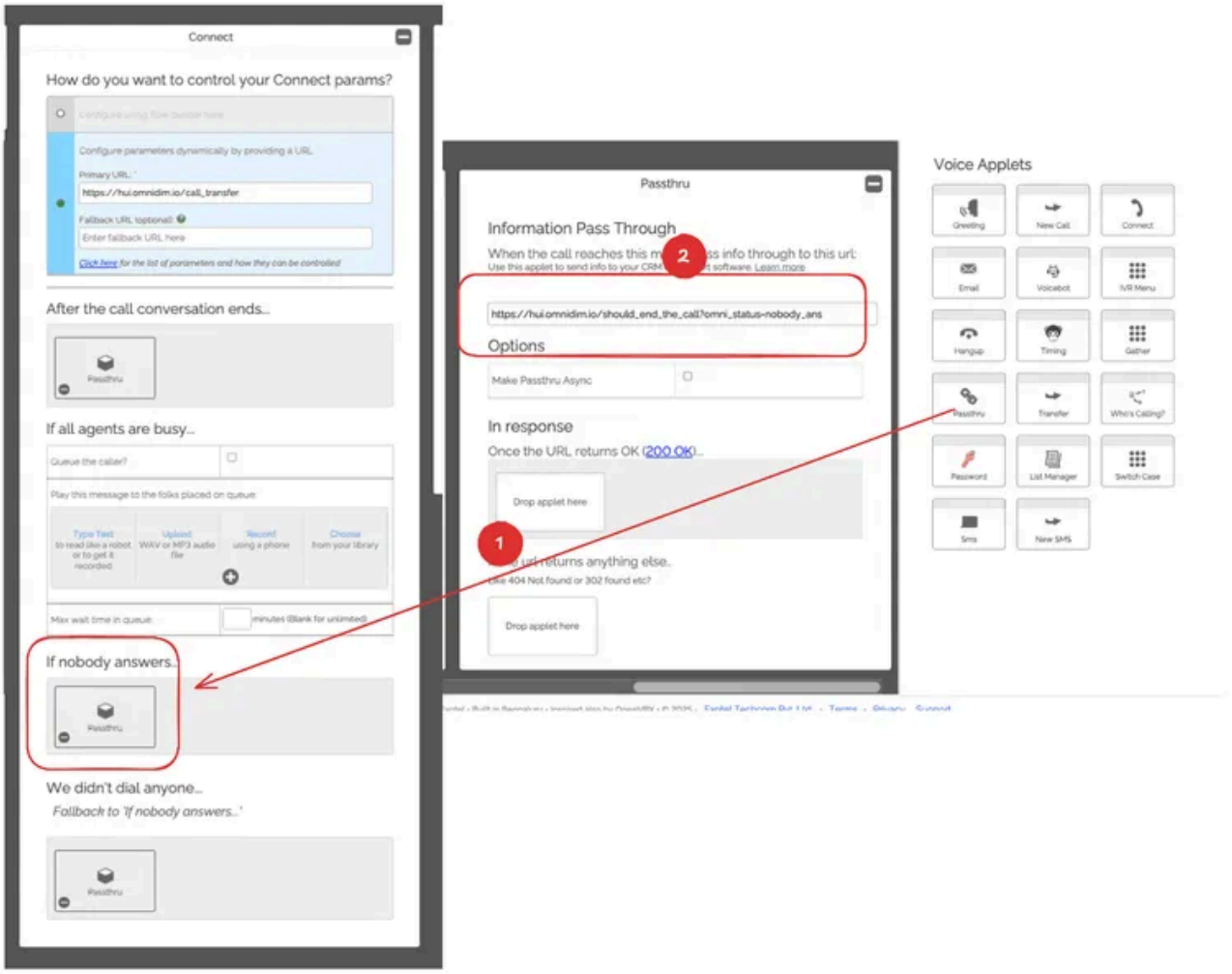
4.8 Add Passthru for Call End

- For the 'After the call conversation ends' option, add a passthru applet with URL: https://live.omnidim.io/should_end_the_call?omni_status=call_end
- Add passthru for call end scenario



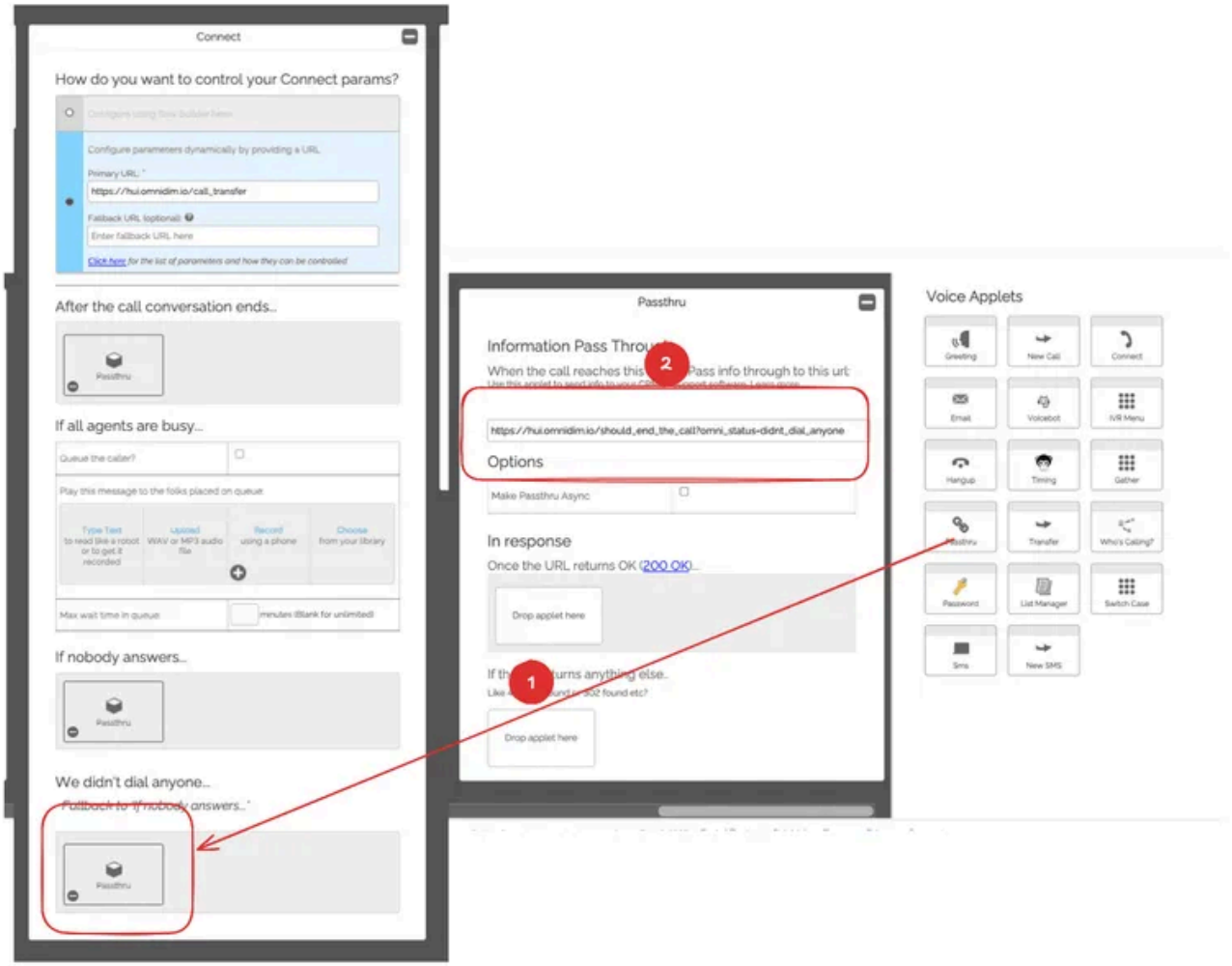
4.9 Add Passthru for No Answer

- For the 'If nobody answers' option, add a passthru applet with URL: https://live.omnidim.io/should_end_the_call?omni_status=nobody_ans
- Add passthru for no answer scenario



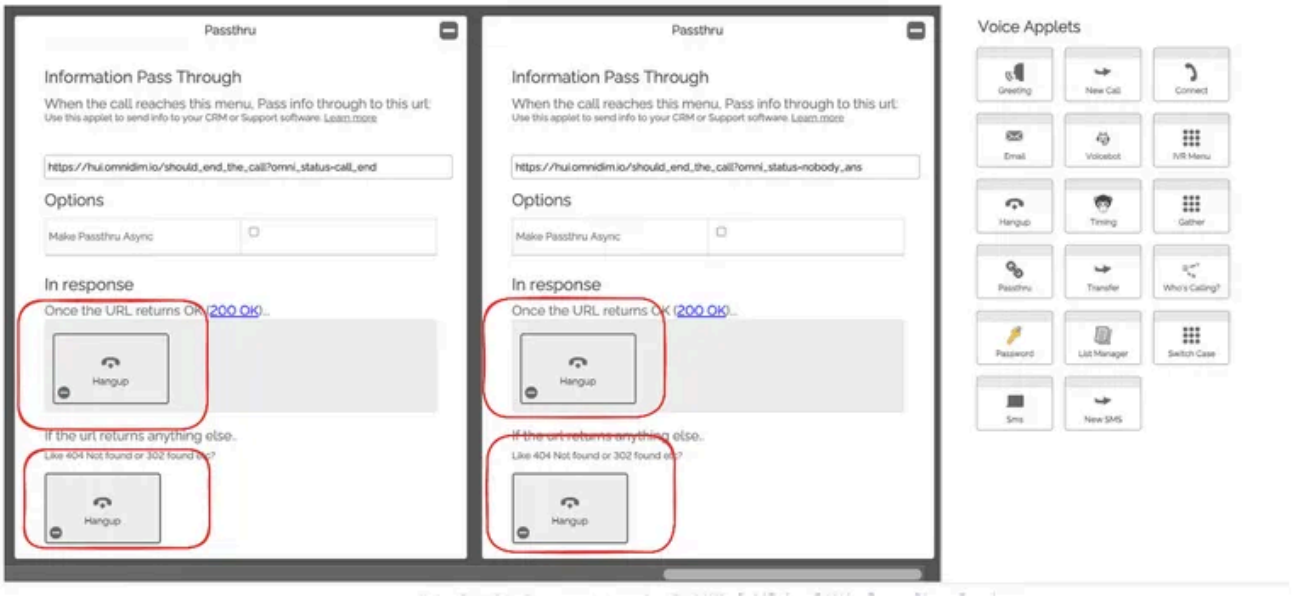
4.10 Add Passthru for No Dial

- For the 'We didn't dial anyone' option, add a passthru applet with URL: https://live.omnidim.io/should_end_the_call?omni_status=didnt_dial_anyone
- Add passthru for no dial scenario



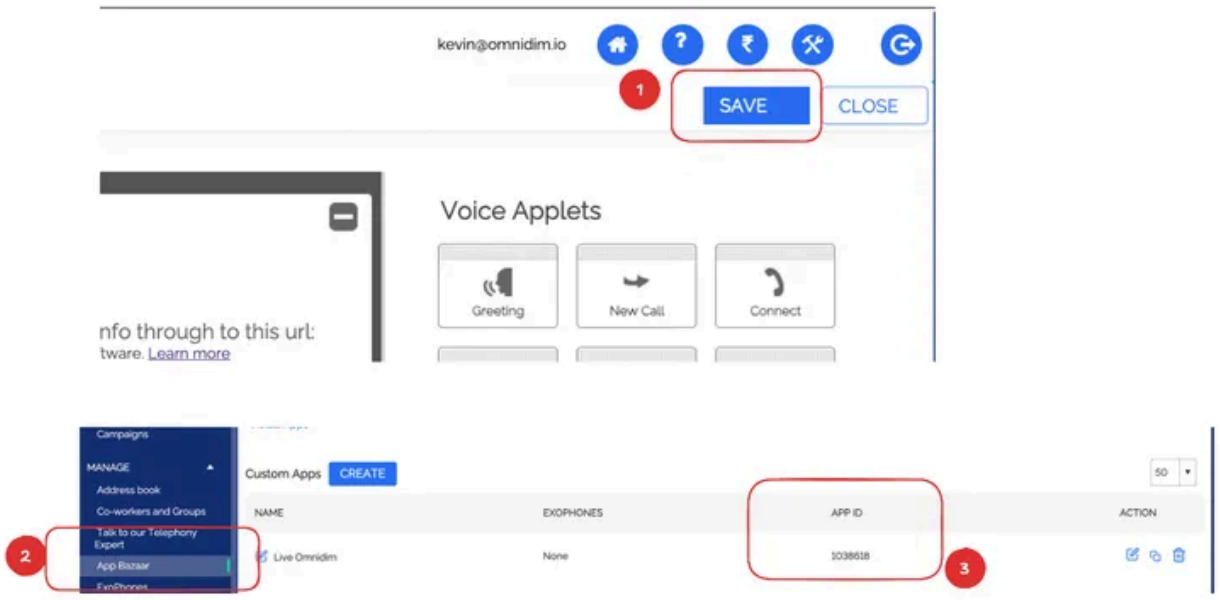
4.11 Add Hangup Applets

- On all the passthru applets you just created, add a 'Hangup' applet in both 200 and 302 responses to ensure calls end properly.
- Add hangup applets to all passthru responses



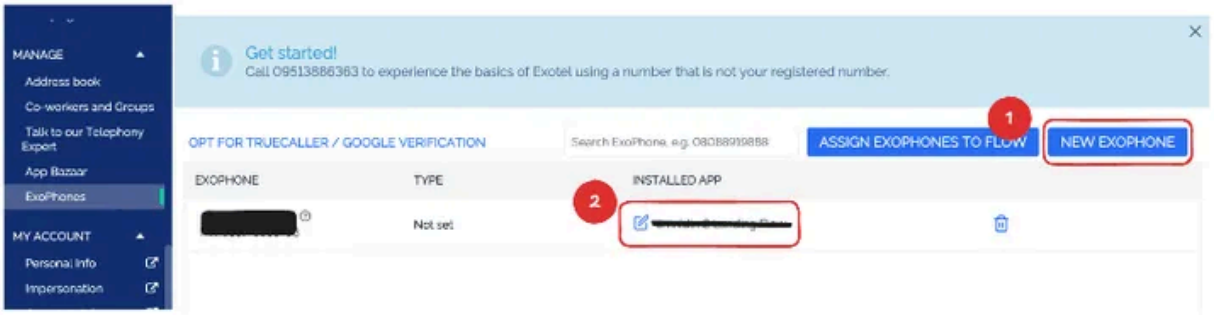
4.12 Save Your Call Flow

- Click 'Save' to save your call flow configuration and close the flow builder. Make sure to note down the App ID (call flow ID) as you'll need it for the import process.
- Save your call flow configuration



4.13 Attach Call Flow with your Exophone number

- Go to the Exophone section in the Exotel dashboard, buy a number and attach the call flow you just created. Make sure to note down the Phone Number as you'll need it for the import process.
- Attach the call flow to your Exophone number.



5. Importing the Number to OmniDim

- Go to Phone Numbers Section
- Log in to OmniDim dashboard → **Phone Numbers**
  Click “Import Exotel Phone Number”
- Opens an import form
  Fill in All Credentials
  - API Key
  - API Token
  - Subdomain
  - Account SID
  - Exophone number
  - App ID

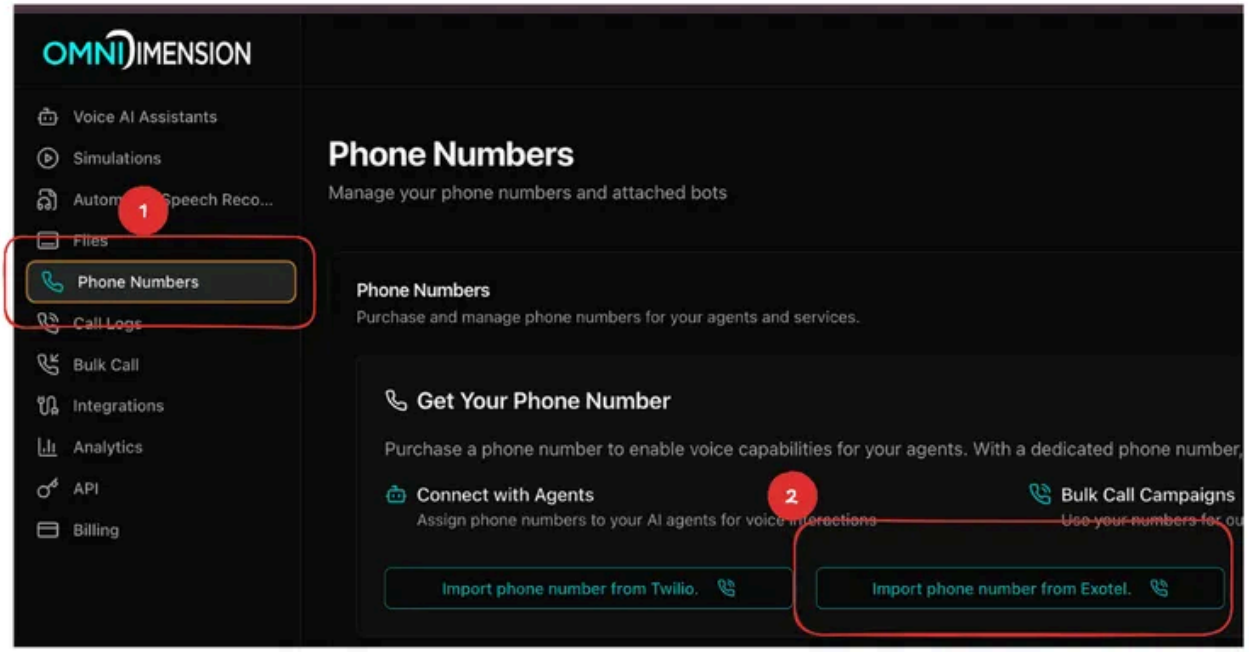
- Submit Import
- Click **Import**
  On success, the phone number will be listed in OmniDim interface

6. Import to OmniDimension

- Now that your call flow is set up in Exotel, follow these steps to import your phone number into OmniDimension:

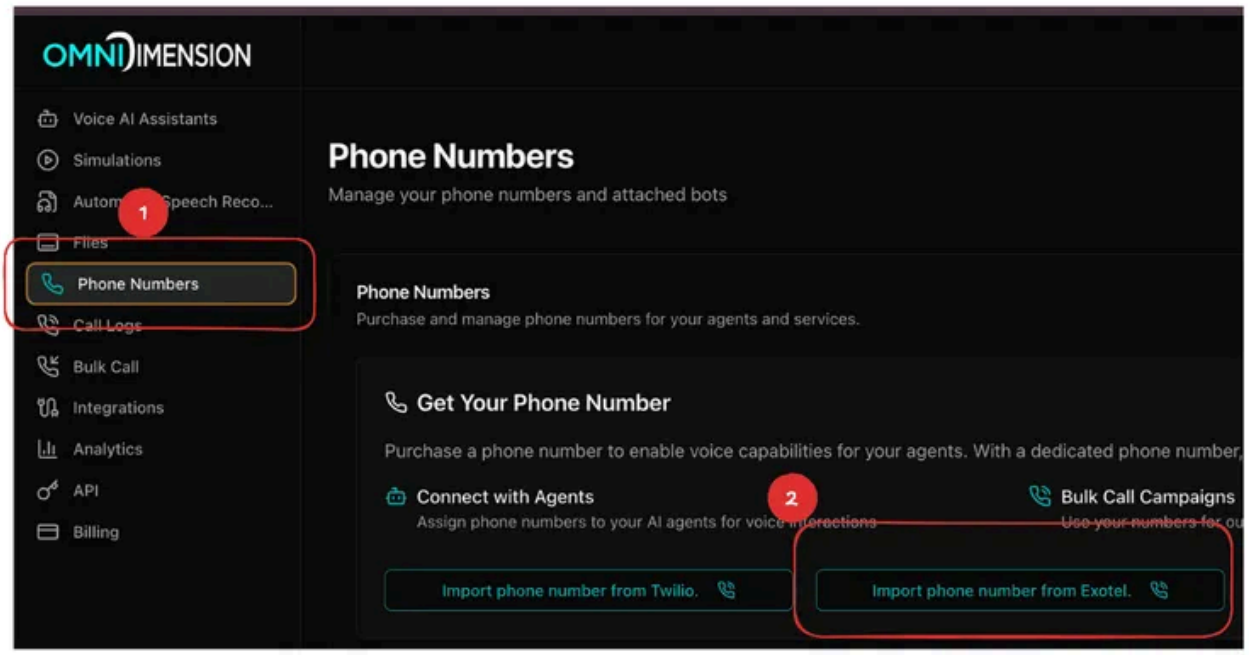
6.1 Navigate to Phone Numbers

- In your OmniDimension dashboard, go to the 'Phone Numbers' section from the main navigation menu.
- Navigate to Phone Numbers in your dashboard



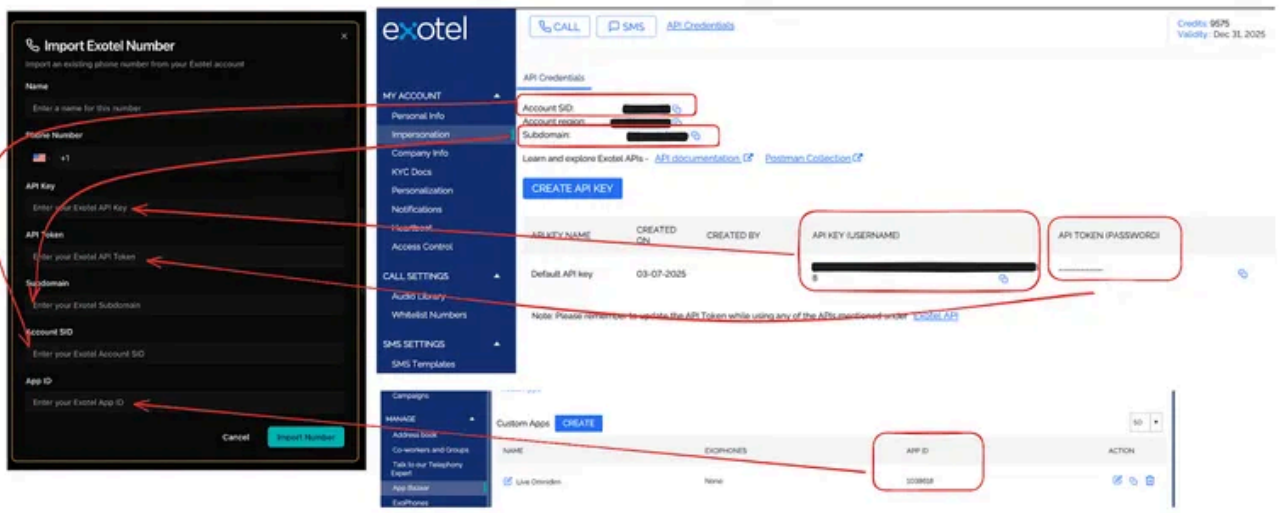
6.2 Click Import Exotel Phone Number

- Look for the 'Import Exotel Phone Number' button and click on it to open the import form.
- Click the Import Exotel Phone Number button



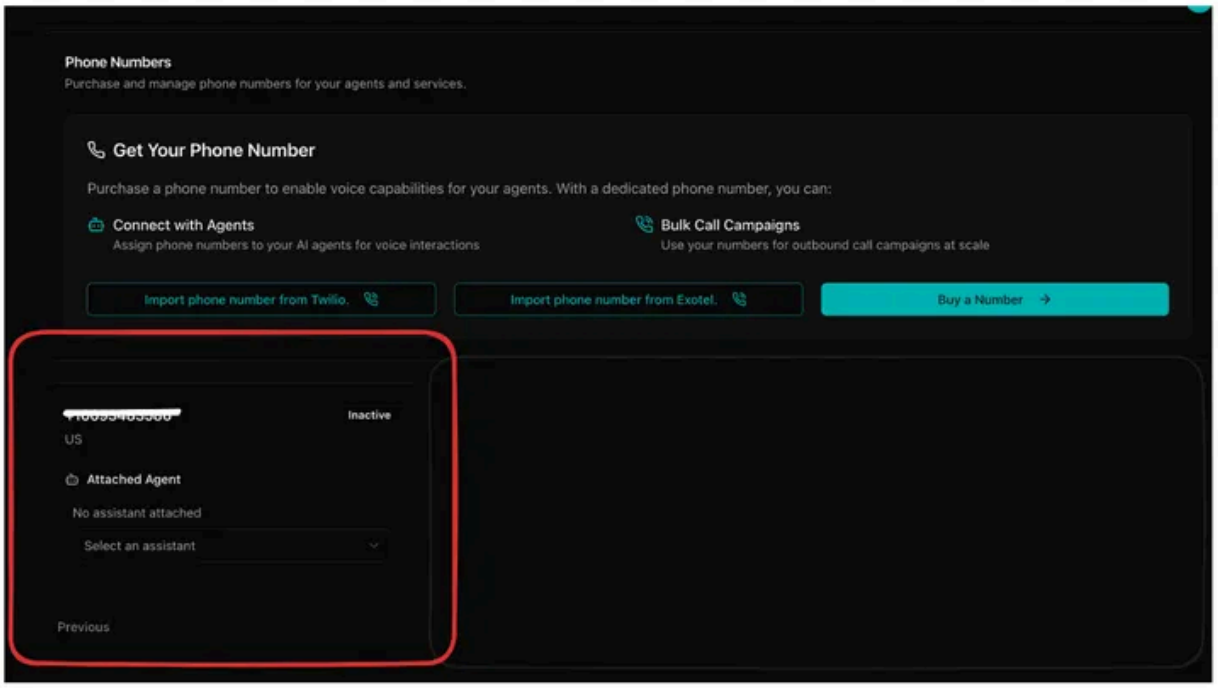
6.3 Fill in the Import Form: Enter all the credentials you collected from your Exotel dashboard in the form fields:

- Exotel API Key
- Exotel API Token
- Exotel Subdomain
- Exotel Account SID
- Exotel Phone Number
- Exotel App ID (your call flow ID)
- Fill in all the required credentials



6.4 Complete the Import

- Click the 'Import' button to complete the process. Your Exotel phone number will now appear in your OmniDimension dashboard and be ready for use with your AI agents.
- Your phone number has now been imported successfully



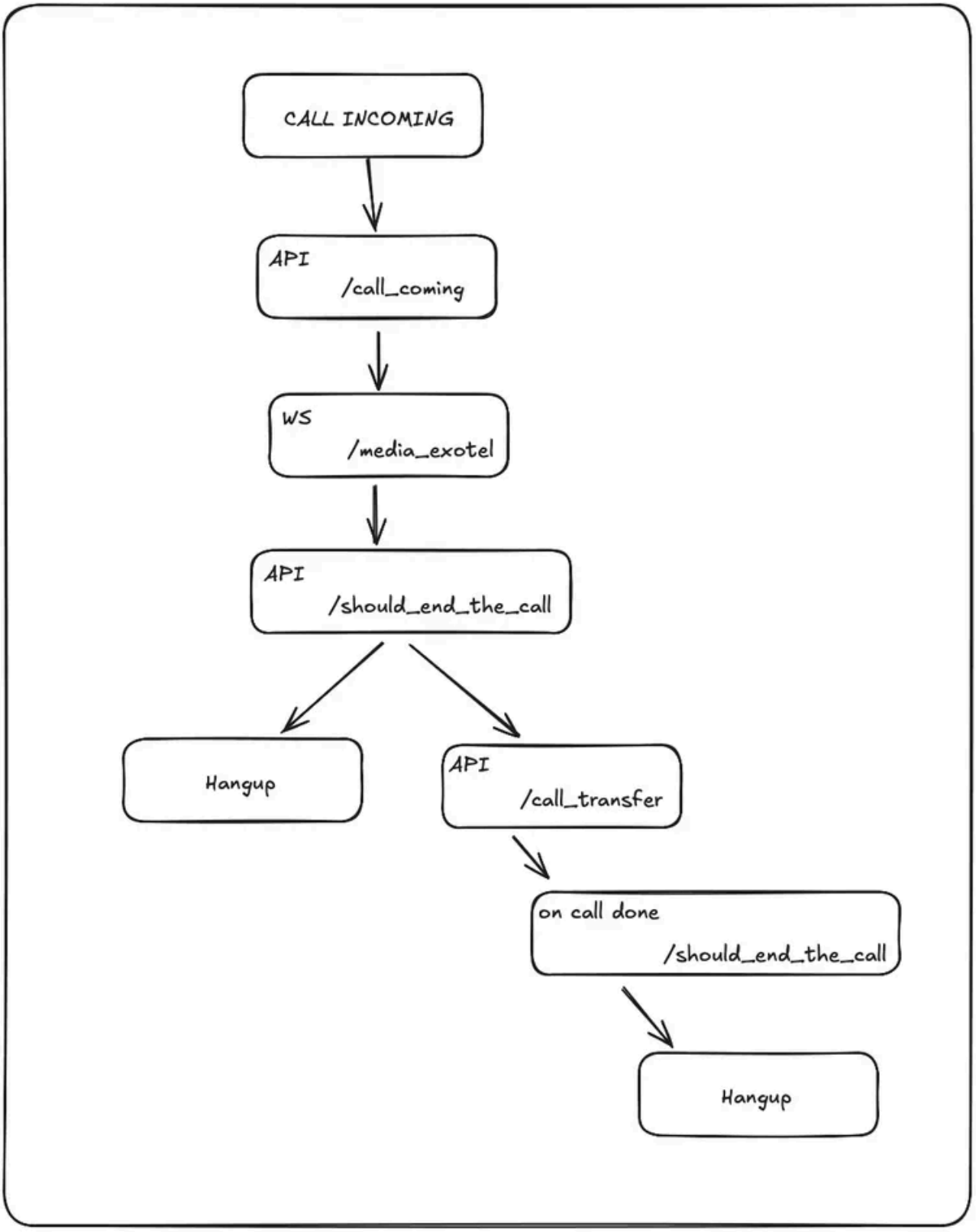
7. Understanding the Call Flow

Here's how your call flow works after setup. When someone calls your Exotel number:

- The call first goes to OmniDimension's call_coming endpoint

- OmniDimension then connects to your AI agent through the voicebot
- Your AI agent handles the conversation
- When the call should end, it goes to the should_end_the_call endpoint
- If a transfer is needed, it goes to the call_transfer endpoint
- All calls eventually end with a hangup action

Complete call flow overview



8. What Happens After Import

Once your phone number is successfully imported, you can:

- Use the phone number with any of your AI agents

- Receive incoming calls that are automatically handled by your agents
- Monitor call logs and performance in your dashboard

9. Next Steps

Now that your Exotel phone number is imported, here are some recommended next steps:

- Configure your AI agents to handle calls effectively
- Attach exotel phone number to your agent
- Test your setup by making a test call to your number

10. Troubleshooting Common Issues

If you encounter any issues during the setup process, here are some common solutions:

- Voicebot applet not visible: Make sure you've completed KYC verification. If still not visible, contact Exotel support
- Import fails: Double-check all credentials are correct and your call flow is properly configured
- Calls not connecting: Verify your call flow URLs are correct and your OmniDimension account is active
- Call hangup on connect: Make sure all URLs are entered exactly as shown with no extra spaces
- KYC issues: Contact Exotel support if you're having trouble with the verification process

11. Getting Help

If you need additional help with the Exotel integration:

- Check the Exotel documentation for detailed call flow setup
- Contact Exotel support for issues with your Exotel account or call flows
- Contact OmniDimension support for issues with the import process or AI agents
- Join our community forums to connect with other users
- Handle retries and timeouts gracefully

10. Support & Resources

- OmniDim support/contact: mitra@omnidim.io
- Exotel Support: hello@exotel.com/808 8919 888