Reference Guide

AudioCodes Intuitive Human Communications for Chatbot Services

Voice.AI Gateway API

Version 2.2

audiocodes

Notice

Date Published: November-04-2020

WEEE EU Directive

Pursuant to the WEEE EU Directive, electronic and electrical waste must not be disposed of with unsorted waste. Please contact your local recycling authority for disposal of this product.

Customer Support

Customer technical support and services are provided by AudioCodes or by an authorized AudioCodes Service Partner. For more information on how to buy technical support for AudioCodes products and for contact information, please visit our website at https://www.audiocodes.com/services-support/maintenance-and-support.

Documentation Feedback

AudioCodes continually strives to produce high quality documentation. If you have any comments (suggestions or errors) regarding this document, please fill out the Documentation Feedback form on our website at https://online.audiocodes.com/documentation-feedback.

Stay in the Loop with AudioCodes

Notes and Warnings

OPEN SOURCE SOFTWARE. Portions of the software may be open source software and may be governed by and distributed under open source licenses, such as the terms of the GNU General Public License (GPL), the terms of the Lesser General Public License (LGPL), BSD and LDAP, which terms are located at https://www.audiocodes.com/services-support/open-source/ and all are incorporated herein by reference. If any open source software is provided in object code, and its accompanying license requires that it be provided in source code as well, Buyer may receive such source code by contacting AudioCodes, by following the instructions available on AudioCodes website.

Related Documentation

Document Name
Voice.AI Gateway Product Description
Voice.AI Gateway Integration Guide
Voice.AI Gateway Security Guidelines
Voice.AI Gateway One-Click Dialogflow Integration Guide
AudioCodes Phone Number Connector

Document Revision Record

LTRT	Description
30940	Initial document release.
30941	Updated to Ver. 1.6: health check added.
30942	Updated to Ver. 1.8: text-to-speech and speech-to-text added; ac-bot-api replaced by ac-api.
30943	Hierarchical document structure modified; CreateConversation replaced by botURL; Configuration section added; Voice.AI Gateway Authentication section renamed Authentication with Bot; Creation of a Conversation section updated; diagram in Conversation Flow updated; Health Check section updated (response)
30944	No updates for Ver. 2.0.
30945	Updated to Ver. 2.2: dialer app API (outbound calling); WebSocket (websocketURL); OAuth 2.0 authentication.

Table of Contents

1 Introduction

AudioCodes Voice.AI Gateway enhances chatbot functionality by allowing human communication with chatbots through voice (voicebot), offering an audio-centric user experience. Integrating the Voice.AI Gateway into your chatbot environment provides you with a single-vendor solution, assisting you in migrating your text-based chatbot experience into a voice-based chatbot.

This document provides AudioCodes Voice.AI Gateway's application programming interface (API):

- Bot API - see Bot API on page 2
- Text-to-speech API - see Text-to-Speech API on page 15
- Speech-to-text API - see Speech-to-Text API on page 17

Prior to reading this document, it's recommended that you read the Voice.AI Gateway Product Description to familiarize yourself with AudioCodes Voice.AI Gateway architecture and solution.

Purpose

This guide provides AudioCodes' APIs for connecting your bot service (proprietary bot or middleware) to AudioCodes Voice.AI Gateway.

Targeted Audience

This guide is intended for developers of bot frameworks and middleware.

2 Bot API

AudioCodes Voice.AI Gateway provides a generic bot API that can be used for connecting it to any bot service that doesn't use the standard bot frameworks (such as Microsoft Azure, Amazon Lex, and Google Dialogflow). This Customer-proprietary bot service could also employ middleware that proxies between it and the Voice.AI Gateway. In such a scenario, it's preferable that the Voice.AI Gateway connects directly to your framework or middleware.

AudioCodes bot API offers the following benefits:

- Easy to implement
- Simple authentication scheme
- Traverses firewalls and HTTP proxies
- Enables the bot to use the Voice.AI Gateway's wide range of features

Overview

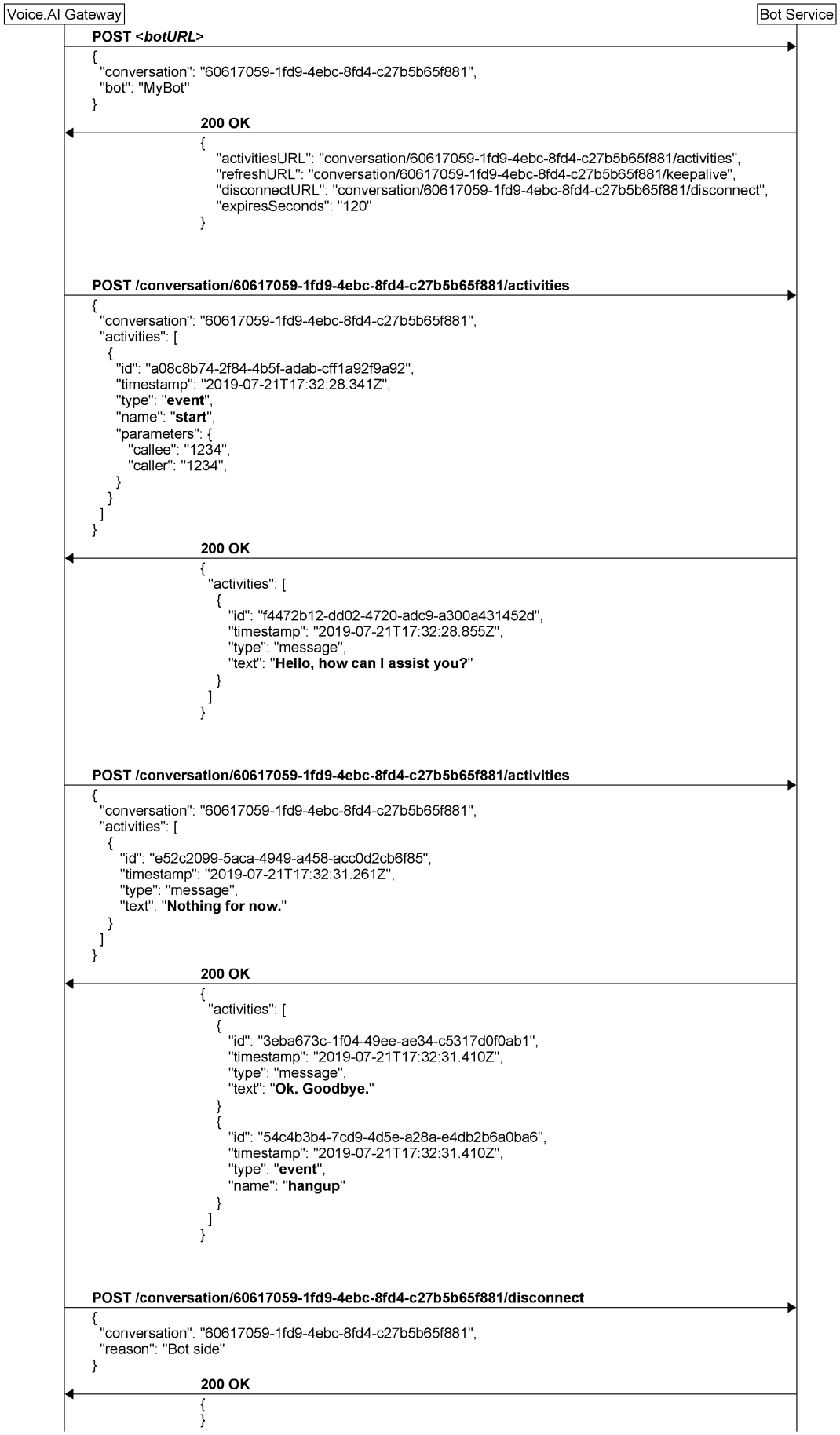
- Roles in the bot API:
  - Client: Voice.AI Gateway
  - Server: Your bot service
- You should implement the server-side of the API so that the Voice.AI Gateway can connect to it.
- The API uses HTTP. All requests by the Voice.AI Gateway are sent to the bot service.
- The API only conveys textual messages (not voice), as the Voice.AI Gateway uses speech-to-Text (STT) and Text-to-Speech (TTS) engines.

Conversation Flow

The conversation flow between the Voice. AI Gateway and the bot service is as follows:

1. The Voice.AI Gateway creates a new conversation by using a pre-configured URL.
2. The reply contains URLs for posting messages to the conversation.
3. Throughout the conversation, the Voice.AI Gateway posts the user's messages to the given URL, while the responses contains the bot's replies.
4. The Voice.AI Gateway ends the conversation.

The following shows an example of a conversation flow between the Voice.AI Gateway and a proprietary bot service:


Voice.AI Gateway
Bot Service
POST <botURL>
{
"conversation": "60617059-1fd9-4ebc-8fd4-c27b5b65f881",
"bot": "MyBot"
}
200 OK
{
"activitiesURL": "conversation/60617059-1fd9-4ebc-8fd4-c27b5b65f881/activities",
"refreshURL": "conversation/60617059-1fd9-4ebc-8fd4-c27b5b65f881/keepalive",
"disconnectURL": "conversation/60617059-1fd9-4ebc-8fd4-c27b5b65f881/disconnect",
"expiresSeconds": "120"
}
POST /conversation/60617059-1fd9-4ebc-8fd4-c27b5b65f881/activities
{
"conversation": "60617059-1fd9-4ebc-8fd4-c27b5b65f881",
"activities": [
{
"id": "a08c8b74-2f84-4b5f-adab-cff1a92f9a92",
"timestamp": "2019-07-21T17:32:28.341Z",
"type": "event",
"name": "start",
"parameters": {
"callee": "1234",
"caller": "1234",
}
}
]
}
200 OK
{
"activities": [
{
"id": "f4472b12-dd02-4720-adc9-a300a431452d",
"timestamp": "2019-07-21T17:32:28.855Z",
"type": "message",
"text": "Hello, how can I assist you?"
}
]
}
POST /conversation/60617059-1fd9-4ebc-8fd4-c27b5b65f881/activities
{
"conversation": "60617059-1fd9-4ebc-8fd4-c27b5b65f881",
"activities": [
{
"id": "e52c2099-5aca-4949-a458-acc0d2cb6f85",
"timestamp": "2019-07-21T17:32:31.261Z",
"type": "message",
"text": "Nothing for now."
}
]
}
200 OK
{
"activities": [
{
"id": "3eba673c-1f04-49ee-ae34-c5317d0f0ab1",
"timestamp": "2019-07-21T17:32:31.410Z",
"type": "message",
"text": "Ok. Goodbye."
}
{
"id": "54c4b3b4-7cd9-4d5e-a28a-e4db2b6a0ba6",
"timestamp": "2019-07-21T17:32:31.410Z",
"type": "event",
"name": "hangup"
}
]
}
POST /conversation/60617059-1fd9-4ebc-8fd4-c27b5b65f881/disconnect
{
"conversation": "60617059-1fd9-4ebc-8fd4-c27b5b65f881",
"reason": "Bot side"
}
200 OK
{
}

API

This section provides the bot APIs.

Before You Begin

Prior to using this API, please note the following:

- All Voice.AI Gateway requests use HTTP POST request methods.
- All requests and responses contain a JSON body and with the appropriate 'Content-Type: application/json' header.
- All JSON bodies must be encoded with UTF-8.
- Any non-200 response is considered a failure and disconnects the conversation. Failure responses can optionally contain a JSON body with a `reason` attribute.
- All requests have a timeout of 20 seconds. If the timeout expires and no response has been received, the conversation is disconnected.
- The Voice.AI Gateway uses connection reuse (HTTP Connection Keep-Alive). It's recommended that the bot service sets the HTTP Keep-Alive time to at least 30 seconds.
- If a connection error occurs, the Voice.AI Gateway retries the request. Note that the Voice.AI Gateway ignores duplicated activity IDs and therefore, retrying is not expected to cause double handling of the activities.

Configuration

The Voice.AI Gateway should be configured with a `botURL` parameter. The Voice.AI Gateway uses the `botURL` to connect to your bot, for two purposes:

- to verify that the URL is valid and active (using HTTP GET, as described in Health Check on page 12)
- to create new conversations (using HTTP POST, as described in Creation of a Conversation on the next page)

For Rasa bots, a typical value for `botURL` has the form: `http://<host>/webhooks/audiocodes/webhook`

Other relevant configuration parameters include:

- Security: Depends on one of the following authentication methods (see Authentication with Bot on page 13):
  - `token`: Static access token used to authenticate the communication with the bot
  - `oauthTokenUrl`: OAuth access token provided by an authorization server for OAuth 2.0 authentication.
- `providerBotName`: This is the value that is sent on the creation of the conversation, to allow one `botURL` to be used for several bots.

Creation of a Conversation

To start a conversation, the Voice.AI Gateway sends a POST request to the `botURL`. The Voice.AI Gateway sends the unique ID of the conversation in the `conversation` attribute. If several bots share the same URL, the Voice.AI Gateway can be configured to add a `bot` attribute to the request body.

The body of the response from the bot service should contain a set of URLs for performing actions on the newly created conversation. The URLs should be unique to the conversation, by containing a UUID as part of the path - either by using the ID from the `conversation` attribute or a UUID generated by the bot service.

If a URL is relative, the Voice.AI Gateway resolves the URL using the `botURL` parameter as the base URL (according to Section 4 of RFC 1808).

After the conversation is created, the Voice.AI Gateway sends an activity with the `start` event. For more information on the start message, refer to the Voice.AI Gateway Integration Guide.

Request Body Attributes

Parameter	Type	Description
`conversation`	String	Voice.AI Gateway's conversation ID.
`bot`	String	(Optional) The value of the `providerBotName` parameter (if exists).
`outboundTarget`	String	Contains the "target" of the dial-out request. **Note:** This is applicable only to the Outbound Call feature. For more information, see Dialer App API for Outbound Calls on page 22.

Response Body Attributes

Parameter	Type	Description
`activitiesURL`	String	Relative or absolute URL. The Voice.AI Gateway sends activities to this URL.
`refreshURL`	String	Relative or absolute URL.
`disconnectURL`	String	Relative or absolute URL.
`websocketURL`	String	Relative or absolute URL. This URL

Parameter	Type	Description
		indicates that the bot (server) supports sending activities using WebSocket. For more information, see Sending Activities over WebSocket on page 9.
`expiresSeconds`	Number	The value can be from 60 to 3600. The recommended value is 120. For more information on conversation refreshes, see Conversation Refresh on page 10.

Example

The following shows an example of creating a conversation:

- Request:

```
{
 "conversation": "ad8f59d2-4a72-4f19-ad34-e7e9b1636111"
}
```

- Response:

```
{
 "activitiesURL": "conversation/ad8f59d2-4a72-4f19-ad34-
e7e9b1636111/activities",
 "refreshURL": "conversation/ad8f59d2-4a72-4f19-ad34-e7e9b1636111/refresh",
 "disconnectURL": "conversation/ad8f59d2-4a72-4f19-ad34-
e7e9b1636111/disconnect",
 "expiresSeconds": 60
}
```

Sending and Receiving Activities

The messages sent between the parties of the conversation are called activities. When the Voice.AI Gateway has activities to send, it sends a POST request to the URL specified in `activitiesURL`. The body of the POST request includes an `activities` attribute containing an array of activities.

The body of the response should also include an `activities` attribute containing an array of activities. If no activities are needed, either the `activities` attribute is omitted or it's sent with an empty array.

If the conversation doesn't exist, the bot service should respond with a 404 Not Found.

The format of the activities is described in the Voice.AI Gateway Integration Guide. In addition, each `activity` must include the following additional attributes:

- `id`: The sender of an activity should generate a UUID (RFC 4122, v4) per activity and send it in the `id` attribute. The receiver of activities should retain a set of all the received activities IDs (in the current conversation) and ignore duplicate activities. This allows the resending of activities in case of failures, without the activities being handled twice.
- `timestamp`: The sender of an activity should add a `timestamp` attribute containing the current time. The format of the timestamp is according to RFC 3339, where the time is in UTC with 3 decimal digits for milliseconds. For example: "2019-04-23T18:25:43.511Z".

  The `timestamp` must include the creation time of the activity and must not be modified if the activity is re-sent.

  The timestamp is mainly used for logging and debugging.

Request Body Attributes

Parameter	Type	Description
`conversation`	String	Voice.AI Gateway's conversation ID.
`activities`	Array of Objects	Array of activities.

Response Body Attributes

Parameter	Type	Description
`activities`	Array of Objects	Array of activities.

Example

The following shows an example of the `start` activity that is sent by the Voice.AI Gateway when a conversation starts (using `activities` endpoint):

- Request:

```
{
 "conversation": "ad8f59d2-4a72-4f19-ad34-e7e9b1636111",
 "activities": [
  {
   "id": "ecf2d78d-ef7b-4a5e-907c-53c97cef5f97",
   "timestamp": "2020-01-26T13:03:48.745Z",
   "type": "event",
```

```
    "name": "start",
    "parameters": {
     "callee": "1234",
     "calleeHost": "10.20.30.40",
     "caller": "+123456789",
     "callerHost": "10.20.30.40"
    }
   }
 ]
}
```


- Response:


```
{
 "activities": [
  {
   "id": "15b3d407-5161-41e7-8114-a273859c5f6d",
   "timestamp": "2020-01-26T13:03:48.748Z",
   "type": "message",
   "text": "Hi there."
  }
 ]
}
```


The following shows an example of `message` activities that correspond to speech utterances:

- Request (to `activitiesURL`):


```
{
  "conversation":"55b77909-82d8-4355-87f1-68081f4dbb36",
  "activities":[
    {
      "id":"bc44c054-846d-490d-85e9-d3aea96b4f0f",
      "timestamp":"2019-08-20T14:09:12.251Z",
      "type":"message",
      "text":"Hi.",
      "parameters":{
        "confidence":0.6599681377410889,
        "recognitionOutput":{
          "RecognitionStatus":"Success",
          "Offset":32300000,
          "Duration":5800000,
          "NBest":[
            {
```

```
            "Confidence":0.6599681377410889,
            "Lexical":"hi",
            "ITN":"Hi",
            "MaskedITN":"Hi",
            "Display":"Hi."
          },
          {
            "Confidence":0.3150425851345062,
            "Lexical":"high",
            "ITN":"high",
            "MaskedITN":"high",
            "Display":"high"
          }
        ]
      }
    }
   }
  ]
}
```


- Response:


```
{
 "activities": [
  {
   "id": "dc4eb401-17f2-436f-80fa-b60156b8a804",
   "timestamp": "2020-01-26T13:04:00.885Z",
   "type": "message",
   "text": "How may I assist you?"
  }
 ]
}
```


Sending Activities over WebSocket

Typically, bots based on the AudioCodes Bot API operate by request- response communication. The user's input is conveyed to the bot in the request, and the bot's immediate response is conveyed in the response. The Asynchronous API allows bots to also send any messages (activities) asynchronously to users through the Voice.AI Gateway (i.e., without requiring a request).

An example of a scenario where this asynchronous feature could be implemented is when the bot needs to perform a time consuming action such as fetching information from a database. In this scenario, the bot may first send a reply of "please wait" to the user, and then once the information is retrieved, sends a message to the user with the information.

Asynchronous messages are sent through a WebSocket connection (over HTTPS). Upon the start of a conversation between the bot and user, the Voice.AI Gateway sends a POST request to the `botURL`. The body of the response from the bot contains a set of URLs for performing actions on the newly created conversation (as described in Creation of a Conversation on page 5). One of these URLs can be the `websocketURL`, which indicates that it supports sending activities using WebSocket. The Voice.AI Gateway then opens a WebSocket connection with this URL (bot acting as the server).

The WebSocket connection can be secured using HTTPS and OAuth 2.0 authorization (Voice.AI Gateway sends an Authorization header on the WebSocket establishment request). If WebSocket connection establishment fails, the conversation is terminated with an error. When the WebSocket connection is established, the bot can send any `activities` through this connection (`websocketURL`) whenever it wants. It can still send synchronous activities in responses to the Voice.AI Gateway requests, using the normal connection instead. The Voice.AI Gateway only uses the WebSocket connection to receive activities from the bot; it doesn't send any messages through this connection. The Voice.AI Gateway maintains the WebSocket connection for the entire duration of the conversation. In other words, WebSocket connections are done per conversation. The Voice.AI Gateway closes the WebSocket connection upon the end of the conversation .

Conversation Refresh

The Voice.AI Gateway refreshes the conversation by sending a refresh request to the conversation at least 30 seconds before the `expiresSeconds` value expires. The `expiresSeconds` time is activated upon the start of conversation or last refresh. The refresh is done by sending a POST request to the URL specified in `refreshURL`.

The `expiresSeconds` value can be updated by the response body.

If the bot service doesn't receive a refresh request before `expiresSeconds` value expires, it should consider the conversation as terminated (with an error condition).

If the conversation doesn’t exist, the bot service should respond with a 404 Not Found.

Request Body Attributes

Parameter	Type	Description
`conversation`	String	Voice.AI Gateway's conversation ID.

Response Body Attributes

Parameter	Type	Description
`expiresSeconds`	Number	If the conversation doesn’t receive a refresh, it's closed after the time specified by this parameter. The value can be from 60 to 3600. The recommended

Parameter	Type	Description
		value is 120.

Example

- Request:

```
{
 "conversation": "ad8f59d2-4a72-4f19-ad34-e7e9b1636111"
}
```

- Response:

```
{
 "expiresSeconds": 60
}
```

Ending a Conversation

The conversation may end due to the following reasons:

- The VoIP call has ended (loss of connection with Voice.AI Gateway, or some failure on the SIP side).
- The bot has disconnected (using the `hangup` event, as described in the Voice.AI Gateway Integration Guide).
- An error has occurred.

For any of the above reasons, the Voice.AI Gateway sends a POST request to the URL specified in `disconnectURL`. The body of the POST request can contain a `reason` attribute. The body of the response should be an empty JSON object. If the conversation doesn't exist, the bot service should respond with a 404 Not Found.

If the conversation expires on the bot service side (i.e., no refresh was done by the Voice.AI Gateway), no explicit message is sent by the Voice.AI Gateway.

Request Body Attributes

Parameter	Type	Description
`conversation`	String	Voice.AI Gateway's conversation ID.
`reason`	String	(Optional) The reason for disconnecting the conversation (free text).

Response Body Attributes

The response body is empty.

Example

- Request:

```
{
 "conversation": "ad8f59d2-4a72-4f19-ad34-e7e9b1636111",
 "reason": "Client Side"
}
```

- Response:

```
{
}
```

Health Check

To validate the connection with the bot without creating a conversation, the bot side should handle GET requests to the `botURL` URL without creating a conversation (as conversations are created by POST requests). When the Voice.AI Gateway is deployed as a Software as a Service (SaaS) cloud service, it uses this health-check endpoint to verify that the `botURL` and token that were provided are correct. Upon success, the bot replies with a 200 OK containing a JSON body having the attributes listed below.

Request Body Attributes

The request body is empty.

Response Body Attributes

Parameter	Type	Description
`type`	String	The value is always `ac-bot-api`.
`success`	Boolean	The value is always `true`.

Example

```
{
 "type": "ac-bot-api",
 "success": true
}
```

Security

TLS Usage

It's recommended that the URLs of the bot service use HTTPS. However, for testing environments, HTTP URLs can be used. In addition, the Voice.AI Gateway can be configured to accept self-signed certificates from the bot service.

Authentication with Bot

It's recommended that the Voice.AI Gateway implement an authentication scheme with the bot. This can be one of the following:

- OAuth 2.0 authentication and authorization. For more information, see OAuth 2.0 Authentication below
- Voice.AI Gateway configured with a permanent token value that is sent in the 'Authorization: Bearer <token>' header for every HTTP request to the bot. This token is used by the bot service to authenticate the Voice.AI Gateway. For more information, see Permanent Token Authentication on the next page.

For environments that don't require this authentication (e.g., when implementing an alternative authentication method), the token can be left without a value, and no 'Authorization' header will be sent.

OAuth 2.0 Authentication

The OAuth 2.0 authorization standard can be used to authenticate the Voice.AI Gateway with a bot's service that implements AudioCodes Bot API.

Upon initial communication with the bot, the Voice.AI Gateway acting as a client, requests an access token from a third-party OAuth 2.0 server (determined by Customer). The authorization server identifies the Voice.AI Gateway by the shared secret key, client ID and optionally, scope (all provided to AudioCodes by the Customer). Therefore, the Voice.AI Gateway must be configured with these values (provided by the Customer), using the following parameters under the `providers` section:

- `oauthTokenUrl` – URL of authentication server
- `oauthClientId` – client_id provided by the API
- `oauthClientSecret` – client_secret provided by the API
- `oauthScope` - scope(s) provided by the API

For example:

```
{
  "name": "my_ac_api",
  "type": "ac-api",
  "botURL": "https://localhost:8083/CreateConversation",
  "ttsUrl": "https://localhost:8043/cognitiveservices/v1",
```

```
 "sttUrl": "wss://localhost:8043",
 "oauthTokenUrl": "https://awebrtcoauth.audiocodes.com/auth...",
 "oauthScope": ["somescope"],
 "credentials": {
  "oauthClientId": "my_ac_api",
  "oauthClientSecret": "b16d2ec0-2b4e-4989-93c1-a59933fa2070"
 },
"botAllowSelfSignedCert": true
}
```

Upon receipt of the access token from the authorization server, the Voice.AI Gateway includes this OAuth access token in each HTTP request (Authorization: Bearer header) that it sends to the AudioCodes Bot API that needs to be accessed at the provider.

The access token is used for all subsequent requests and calls by the specific bot, until the token expires. Prior to expiry (about 30 seconds before), the Voice.AI Gateway requests a new access token from the authorization server. If the token expires and no new access token is obtained from the server, existing calls are terminated and no new calls can be made until a new token is obtained.

Permanent Token Authentication

The Voice.AI Gateway can be configured (`credentials` > `token` parameter) with a permanent token value that is sent in the 'Authorization: Bearer <token>' header for every HTTP request. This token is used by the bot service to authenticate the Voice.AI Gateway.

- To prevent malicious attackers from obtaining the token key and accessing resources, store the token key in a secure place.
- If you have configured OAuth 2.0 authentication (as described in OAuth 2.0 Authentication on the previous page) and a permanent token as described in this section, OAuth authentication takes precedence (i.e., permanent token method is ignored).

3 Text-to-Speech API

A third-party TTS vendor (excluding the commonly used ones such as Azure, AWS, and Google) can integrate with an API that the Voice.AI Gateway exposes as a TTS client. The client (Voice.AI Gateway) sends an HTTP POST request towards a pre-defined URL.

An Authorization header is sent by the client on the HTTP request, containing a shared token. The token can be used by the TTS server to identify the client. Example:

```
Authorization: Bearer <token>
```

Request Body Attributes

Parameter	Type	Description
`language`	String	Defines the BCP-47 language code for speech recognition of the supplied audio.
`format`	String	Defines the format of the audio file: ■ `raw`: audio without headers ■ `wav`: audio with WAV headers
`encoding`	String	Defines the manner in which the audio is stored and transmitted. Currently, only 16-bit linear pulse-code modulation (PCM) encoding (`LINEAR16`) is supported.
`sampleRateHz`	Number	Defines the sample rate (in Hertz) of the supplied audio. Currently, only 16,000 Hz is supported.
`voice`	String	Defines the name of the voice used for speech synthesis.
`type`	String	Defines the type of text. If it contains SSML, the type is set to `ssml`.
`text`	String	Defines the text to synthesize.

Response Body Attributes

In case of a success, the TTS server replies with a 200 OK response, containing a body with the synthesized speech. In case of failure, the server replies with an HTTP error code.

Example

- Example 1:

```
{
 "language": "en-US",
 "format": "wav",
 "encoding": "LINEAR16",
 "sampleRateHz": 16000,
 "voice": "SomeVoiceName",
 "text": "Text to be played"
}
```

- Example 2:

```
{
 "language": "en-US",
 "format": "wav",
 "encoding": "LINEAR16",
 "sampleRateHz": 16000,
 "voice": "SomeVoiceName",
 "type": "ssml",
 "text": "<speak><say-as interpret-as=\"ordinal\">1</say-as></speak>"
}
```

4 Speech-to-Text API

The API is based on WebSocket. The client (Voice.AI Gateway) opens a WebSocket connection towards a pre-defined URL, for each conversation. The connection remains open for the whole duration of the conversation. The same connection can be used for several sequential (not concurrent) recognition-sessions on which the STT engine performs recognition of an audio stream. Note that the connection might remain open on periods on which there is no active recognition-session.

Control messages are sent as textual JSON messages. Media is sent as binary frames (per Section 5.6 of RFC 6455). As there is no "session" information on the binary frames, a single WebSocket connection can only be used for one concurrent recognition-session.

Control Messages Sent by Client (Voice.AI Gateway)

- `start` – start a recognition-session.

  Example:

```
{
  "type": "start",
  "language": "en-US",
  "format": "raw",
  "encoding": "LINEAR16",
  "sampleRateHz": 16000,
}
```

  Parameters:

  - `language`: a BCP-47 language code for speech recognition of the supplied audio.
  - `format`: the format of the audio file:
    - `raw`: audio without headers
    - `wav`: audio with WAV headers
  - `encoding`: how the audio is stored and transmitted. Currently, only 16-bit linear pulse-code modulation (PCM) encoding (`LINEAR16`) is supported.
  - `sampleRateHz`: the sample rate (in Hertz) of the supplied audio. Currently, only 16,000 Hz is supported.

  Additionally, if the STT engine supports a "context" for the recognition (e.g., custom vocabulary), a parameter can be sent by the client to indicate the context.

- `stop` – stop the current recognition session.

"stop" can only be sent for a "started" recognition-session.

Example:

```
{
  "type": "stop",
}
```

Control Messages Sent by Server (STT)

- `started` – sent to indicate that the recognition-session has started and that the stream (binary messages) can be sent by the client.

  Example:

  ```
  {
    "type": "started"
  }
  ```

- `hypothesis` – sent for partial recognition.

  Example:

  ```
  {
    "type": "hypothesis",
    "alternatives": [
      {
        "text": "Hi"
      }
    ]
  }
  ```

- `recognition` – sent for each utterance that is recognized by the STT.

Several "recognition" messages can be sent for a single recognition-session.

Example:

```
{
  "type": "recognition",
  "alternatives": [
    {
      "text": "Hi there",
      "confidence": 0.8355
    }
  ]
}
```

- `end` – Sent to indicate that the current recognition-session has ended.

- In case only a single utterance is recognized per recognition-session, an "end" message must be sent immediately after the "recognition" message.
- The "end" message must be sent after the server has received a "stop" message, to indicate that the recognition-session has ended.

Example:

```
{
 "type": "end",
 "reason": "some reason"
}
```

- `error` – indicates that the current recognition-session has ended with a failure condition.

- The same WebSocket can be used to start additional recognition sessions.
- In case of an error that prevents the server from further handling messages on the WebSocket connection, the server must close the connection.

Example:

```
{
 "type": "error",
 "reason": "some error"
}
```

Binary Messages Sent by Client

The client sends the audio stream as WebSocket binary messages, according to the encoding and sample-rate indicated in the `start` message.

Authentication

An Authorization header is sent by the client on the HTTP request that creates the WebSocket connection, containing a shared token. The token can be used by the server to identify the client. For example:

```
Authorization: Bearer <token>
```

Example Flow

Direction	Message

Client > Server	<pre>{
	"type": "start",
	"language": "en-US",
	"format": "raw",
	"encoding": "LINEAR16",
	"sampleRateHz": 16000
	}</pre>

Server > Client	<pre>{
	"type": "started"
	}</pre>

Client > Server	<binary messages>

Server > Client	<pre>{
	"type": "hypothesis",
	"alternatives": [
	{
	"text": "Hi"
	}
	]
	}</pre>

Server > Client	<pre>{
	"type": "recognition",
	"alternatives": [
	{
	"text": "Hi there.",
	"confidence": 0.8355
	}
	]
	}</pre>

Server > Client	<pre>{
	"type": "hypothesis",
	"alternatives": [</pre>

Direction	Message

	<pre>{
	"text": "My name"
	}
	]
	}</pre>

Server > Client	<pre>{
	"type": "recognition",
	"alternatives": [
	{
	"text": "My name is John.",
	"confidence": 0.83
	}
	]
	}</pre>

Client > Server	<pre>{
	"type": "stop"
	}</pre>

Server > Client	<pre>{
	"type": "end",
	"reason": "stop by client"
	}</pre>

	<new recognition-session; same connection>

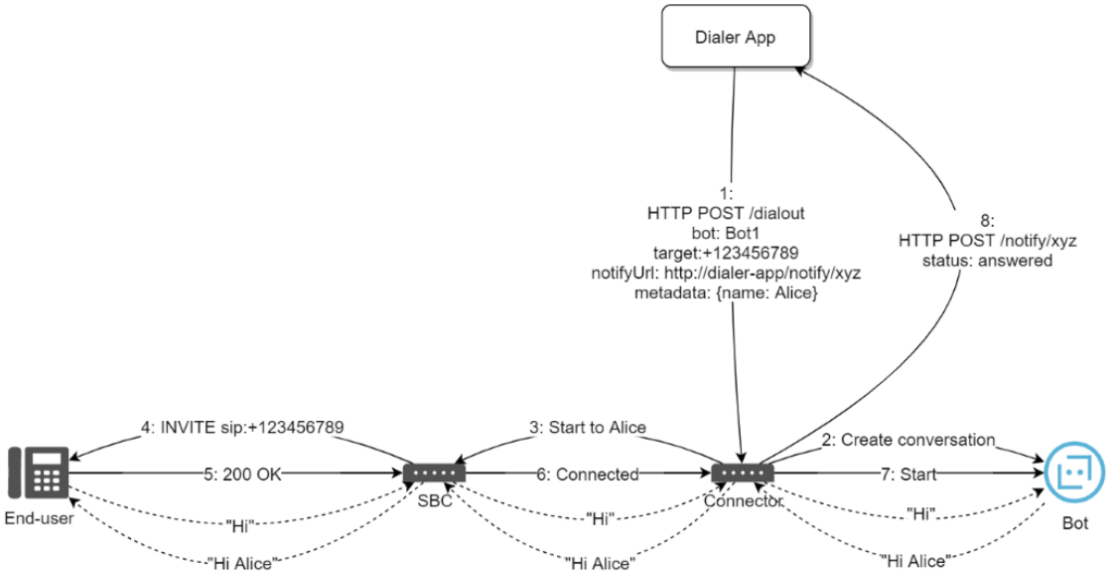
Client > Server	<pre>{
	"type": "start",
	"language": "en-US",
	"format": "raw",
	"encoding": "LINEAR16",
	"sampleRateHz": 16000
	}</pre>

4 Dialer App API for Outbound Calls

In a typical bot deployment, the Voice.AI Gateway first receives the call from the SBC and then sends it to the bot. However, the Voice.AI Gateway can also be triggered by a third-party dialer application to initiate the call instead of the SBC. This setup is often used by companies for outbound dialing campaigns, whereby the dialer automatically initiates calls with potential customers through their bots.

The basic call flow for outbound dialing is as follows:

1. The Voice.AI Gateway receives a message over HTTP (POST) from an outbound dialer application, which triggers it to make an outbound call.
2. The Voice.AI Gateway establishes a connection with the bot.
3. The Voice.AI Gateway initiates a call (outbound) to the user (e.g., PSTN number) through the SBC.
4. When the user answers the call, STT begins and the user's first utterance (e.g., "Hi") is sent to the bot to trigger its logic (e.g., "Hi there, we are calling from…").
5. This feature also allows the Voice.AI Gateway to send (HTTP POST requests) the dialer application asynchronous status notifications of the call (e.g., when the call is answered or failed).



Triggering Outbound Calls

The dialer application triggers the Voice.AI Gateway to make an outbound call (*dial-out*), by sending an HTTP POST request to an endpoint exposed by the Voice.AI Connector (e.g., api/v1/actions/dialout).

Authentication is done using OAuth 2.0, by requesting an access token from the bot. An Authorization header is then sent by the dialer application (client) in the HTTP request, containing the shared token. For example:

```
Authorization: BEARER 223d0d870a364a4a
```

The Voice.AI Gateway grants the client permission to perform a dial-out only if the token is correct for the dial-out OAuth scope (as configured on the Voice.AI Gateway).

Request Body Attributes (JSON)

Parameter	Type	Description
`bot`	String	Name of the bot that must be connected to the conversation. The value is configured on the Voice.AI Gateway using the `providerBotName` parameter.
`target`	String	URI of the target of the outbound call. The Voice.AI Gateway forwards this URI to the SBC and is used for routing (i.e., used as the DestURI). For example: “tel:123456789”.
`caller`	String	User-part of the caller-id of the outbound call.
`callerHost`	String	(Optional) Host-part of the caller-id of the outbound call. If omitted, it is filled with an arbitrary value.
`callerDisplayName`	String	(Optional) Display name of the caller-id of the outbound call.
`notifyUrl`	String	(Optional) URL address of the dialer application to where the Voice.AI Gateway sends notifications. If not provided, no notifications are sent.
`metadata`	JSON Object	(Optional) Data sent to the bot (provides bot information about the call such as the name of the target).
`answerTimeoutSec`	Number	(Optional) Number of seconds to wait for the end-user to answer.

For example:

```
{
 "bot": "Bot1",
 "target": "tel:+123456789",
 "caller": "1-800-111-111",
 "callerHost": "example.com",
```

```
 "callerDisplayName": "My company",
 "answerTimeoutSec": 20,
 "notifyUrl": "https://my-app/call/454/notify",
 "metadata": {
  "participantName": "Alice"
 }
}
```

Response Body Attributes (JSON)

The Voice.AI Gateway responds with a 200 OK after successfully connecting to the bot (before dialing to the user through the SBC).

Parameter	Type	Description
`conversationId`	String	UUID of the newly created conversation.

For example:

```
{
 "conversationId": "daf0c30f-e7a7-4644-b20b-667676b70615"
}
```

In error conditions, Voice.AI Gateway responds with a 500 code.

Status Notifications for Dialer Application

The Voice.AI Gateway can send the dialer application asynchronous status notifications of the call (e.g., when the call is answered or fails). If the request from the dialer application includes the `notifyUrl` parameter, the following notifications are sent by the Voice.AI Gateway to the dialer application URL address specified by `notifyUrl`:

- "answered": When the user answers the call.
- "completed": When the call ends after the user answered it.
- "failed": When the call ends before the user has answered it.

The notifications are sent by the Voice.AI Gateway using HTTP POST with a JSON body consisting of the following parameters:

Parameter	Type	Description
`conversationId`	String	UUID of the conversation (matches the conversationId sent in the response from the Dialer App).

Parameter	Type	Description
`status`	String	Call status: ■ "answered" ■ "failed" ■ "completed"
`reason`	String	If the status is "failed", provides the reason for the failure: ■ "no-answer": There was no answer to the call (due to expiry of `answerTimeoutSec` or due to a SIP timeout). ■ "busy": The target number was busy. ■ "declined": The end-user declined or rejected the call. ■ "error": An error occurred before the end-user answered the call (for example, the target number was invalid).
`reasonText`	String	Free text describing the reason for the failure (for example, the SIP Reason header).

The response to the HTTP POST should be a 200 OK without a body.

An example of a request body:

```
{
 "conversationId": "daf0c30f-e7a7-4644-b20b-667676b70615",
 "status": "failed",
 "reason": "busy",
 "reasonText": "SIP ;cause=486 ;text=\"Busy Here\""
}
```

No authentication is specifically defined for notification requests (URL may include a token/id for authentication). The dialer application can use the `conversationId` to identify the corresponding operation. Alternatively, the dialer application can set a unique `notifyUrl` per operation and identify the conversation according to the URL.

This page is intentionally left blank.

International Headquarters

1 Hayarden Street,

Airport City

Lod 7019900, Israel

Tel: +972-3-976-4000

Fax: +972-3-976-4040

AudioCodes Inc.

200 Cottontail Lane

Suite A101E

Somerset NJ 08873

Tel: +1-732-469-0880

Fax: +1-732-469-2298

Contact us: https://www.audiocodes.com/corporate/offices-worldwide

Website: https://www.audiocodes.com/

Documentation Feedback: https://online.audiocodes.com/documentation-feedback

Document #: LTRT-30945