Conversational AI Agent v0.3 Global Quick Start

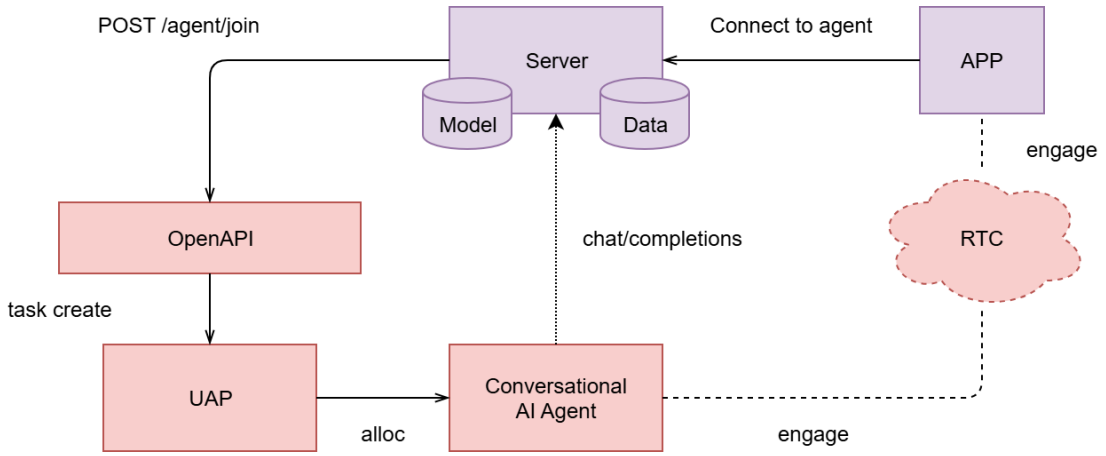
Conversational AI Agent is a next-generation AI Agent solution provided by Agora, designed to help clients simplify real-time interactions with intelligent agents.

Currently, this service is in the Private Beta stage, and the final agreement and API might undergo slight adjustments; your understanding is appreciated.

At present, the service capabilities need to be manually activated. For activation details, please contactshensiwei.

The difference will be marked in **RED** with last version:Conversational AI Agent v0.2 Global Quick Start

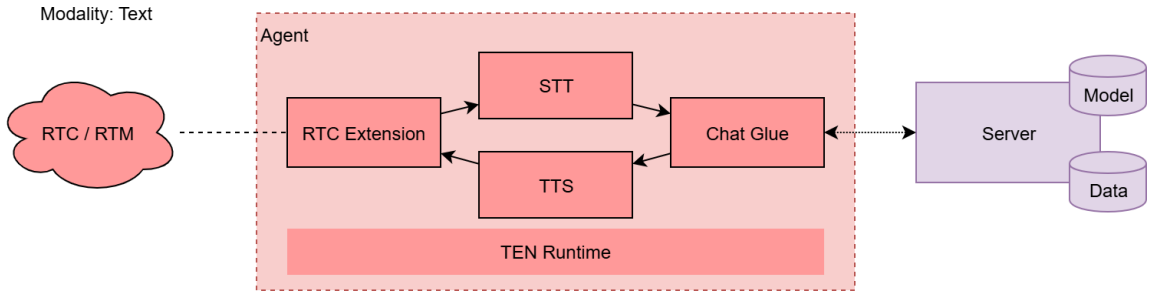
Architecture



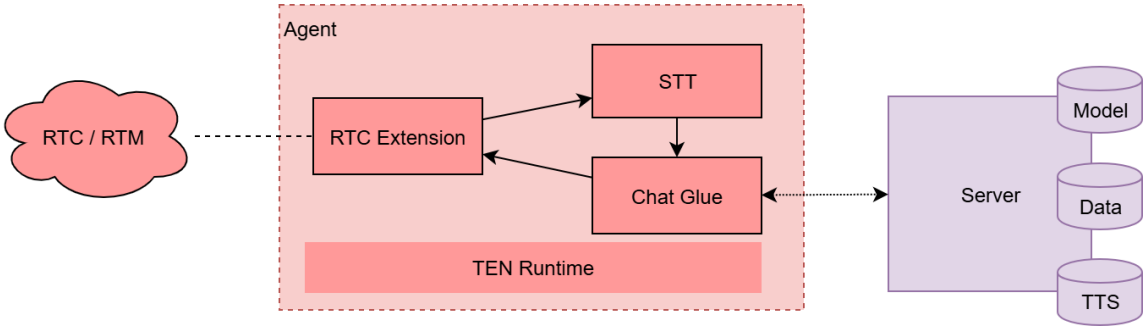
Through the RESTful API provided by Agora, clients can allocate an AI Agent to join a specified RTC channel for 1-on-1 communication with end user.

Note: Does not support multi-user communication.

The Conversational AI Agent includes STT (Speech-to-Text) and TTS (Text-to-Speech) modules. It makes requests to the client's provided URL viaan OpenAI-compatible protocoland retrieves the response in a streaming. The detailed internal implementation is as follows:



For STT and TTS, they are provided by Agora. If clients need to use their own implemented or integrated TTS, they can opt to use the Audio Modal supported by Chat Glue. The specific implementation is as follows:



In this scenario, the input is text, and the output is audio. Chat Completion directly supportsthe audio protocolused in OpenAI's audio-preview, accepting base64-encoded PCM segments.

API

Agent Join

Method: POST

URL: https://api.agora.io/api/conversational-ai-agent/v1/projects/{appid}/join

Parameters:

- Name: Unique identifier for the Agent, identical identifiers cannot be created repeatedly
- Properties: Specific parameters for the Agent, details are as follows

```
{
    "channel": "channel_name",
    "token": "token",
    "agent_rtc_uid": 1234,
    "remote_rtc_uid": 123,
    "input_modalities": ["text"],
    "output_modalities": ["text"],
    "idle_timeout": 120,
    "audio_scenario": 9,
    "rtc_codec": -1,
    "custom_llm": {
        "url": "https://api.openai.com/v1/chat/completions",
        "token": "sk-xxx",
        "prompt": "You are a helpful chatbot.",
        "model":"gpt-4o-mini",
        "greeting":"Merry Christmas, how can I assist you today?",
        "failure_message":"Please hold on a second.",
        "extra_metadata": {
            "userName": "Tomas"
        },
        "metadata_enabled": true
    },
    "tts": {
        "voice_id": "en-US-AndrewMultilingualNeural"
    },
    "asr": {
        "language": "en-US"
    },
    "vad": {
        "silence_duration_ms": 1000
    }
}
```

Descriptions for the parameters:

Property	Type	Description	Default Value	Notes
token	string	rtc channel token	""	
channel	string	rtc channel name	Required	
agent_rtc_uid	int32	rtc int user id	Required	0: random user id [do not forget token] other: specific int user id for agent
remote_rtc_uid	int32	rtc subscription id	Required	0: subscribe all users in channel other: subscribe single user with specific int user id

rtc_codec	**int32**	**agent publish codec**	**-1**	**0: PCMU** **8: PCMA** **9: G722** **120: OPUS** **122: OPUSFB**
audio_scenario	**int32**	**audio scenario**	**0**	**0: default** **9: AI QOS (only support native sdk with AI QOS)**
input_modalities	array[string]	input modalities for custom llm	["text"]	**Suport ["text"],["text", "image"]**
output_modalities	array[string]	output modalities for custom llm		Support ["audio"], ["text"],["text", "audio"]
idle_timeout	int	max idle seconds for agent	30	0: agent only stop for manual leave
asr.language	string	asr language	“en-US”	zh-CN, en-US
tts.voice_id	string	tts voice id	"en-US-AndrewMultilingualNeural"	For supported voice IDs, refer to the appendix
custom_llm.url	string	callback url for custom llm	Required	
custom_llm.token	string	verify token for custom llm	""	Must active token in Production Env
custom_llm.prompt	string	prompt for custom llm	“”	System prompt
custom_llm.model	string	model id for custom llm	""	
custom_llm.max_history	**int32**	**short term memory length for custom llm**	**10**	**0: no short term memory on agent side** **User and Assistant message will be count seperately**
custom_llm.greeting	string	greeting message	""	greet the first rtc user who subscibed for joining the channel
custom_llm.failure_message	string	failure message	""	read out when custom llm call fails
custom_llm.extra_metadata	dict	metadata info	{}	passed to the server via the `metadata` parameter with each custom llm request
custom_llm.metadata_enabled	bool	flag for bypassing metadata	false	*Noted that none of the third-party llm service will support this.* *Enable only when you are hosting your own custom llm service.*
vad.interrupt_threshold	float	interruption sensitivity	0.8	0.0 - 1.0
vad.prefix_padding_ms	int	prefix padding in millisecond	300	
vad.silence_duration_ms	int	slience_duration in millisecond	500	
vad.threshold	float	vad sensitivity	0.3	0.0 - 1.0

Success

StatusCode: 200

Property	Type	Description
agent_id	string	A unique identifier for the instance, used for subsequent deletion & get.
create_ts	int	create timestamp

state	string	current status

Failed

StatusCode：non-200

Property	Type	Description
detail	string	Detailed failure info
reason	string	Failure reason

For error codes, refer to the appendix.

Agent Leave

Method: POST

URL:https://api.agora.io/api/conversational-ai-agent/v1/projects/{appid}/agents/{agentId}/leave

The agentId should be filled in with the agent_id returned by the join operation.

Agent Update

Allow parameters change for in-progress agent

Method: POST

URL:https://api.agora.io/cn/api/conversational-ai-agent/v1/projects/{appid}/agents/{agentId}/update

The agentId should be filled in with the agent_id returned by the join operation, only the following parameters are allow to update:

Property	Type
token	string
tts.voice_id	string
custom_llm.url	string
custom_llm.token	string
custom_llm.prompt	string
custom_llm.model	string
custom_llm.greeting	string
custom_llm.failure_message	string
custom_llm.extra_metadata	dict
custom_llm.metadata_enabled	bool
vad.interrupt_threshold	float
vad.prefix_padding_ms	int
vad.silence_duration_ms	int
vad.threshold	float

Agent Get

Get active agent's status

Method: GET

URL:https://api.agora.io/cn/api/conversational-ai-agent/v1/projects/{appid}/agents/{agentId}

Agent List

Fetch agents info that meets the filter

Method: GET

URL:https://api.agora.io/cn/api/conversational-ai-agent/v1/projects/{appid}/agents

Glue Sample

Currently, the request requires the client to prepare the Glue URL and Token themselves.

Third-party LLM Service

Currently, Glue's protocol is compatible with OpenAI, so all large model providers compatible with OpenAI can be directly integrated. For example, to access OpenAI Chat Completion directly, you can use the following method:

```
curl --location 'https://api.agora.io/api/conversational-ai-agent/v1/projects/{appid}/join'\
--header 'Authorization: Basic ${key}'\
--header 'Content-Type: application/json'\
--data '{"name":"unique_name","properties":{"channel":"test_channel", "agent_rtc_uid":1234,
"custom_llm":{"url":"https://api.openai.com/v1/chat/completions", "token":"sk-xxxxx", "model":"gpt-4o-
mini", "prompt":"You are a helpful assistant.", "greeting":"Hi, how can I assist you today?"}}}'-i
```

DIY

The overall protocol is consistent withOpenAI. Below is a sample code in Python:

```
importos
importjson
importtraceback
importlogging
importasyncio

fromtyping importList, Union, Dict, Optional
frompydantic importBaseModel, HttpUrl

fromfastapi.security importHTTPBearer, HTTPAuthorizationCredentials
fromfastapi.responses importStreamingResponse
fromfastapi importDepends, FastAPI, HTTPException, Request

logger =logging.getLogger(__name__)

app =FastAPI(title="Chat Completion API",
             description="API for streaming chat completions with support for text, image, and audio
content",
             version="1.0.0")

classTextContent(BaseModel):
    type: str="text"
    text: str

classImageContent(BaseModel):
    type: str="image"
    image_url: HttpUrl

classAudioContent(BaseModel):
    type: str="input_audio"
    input_audio: Dict[str, str]

classToolFunction(BaseModel):
    name: str
    description: Optional[str]
    parameters: Optional[Dict]
```

```
    strict: bool=False

classTool(BaseModel):
    type: str="function"
    function: ToolFunction

classToolChoice(BaseModel):
    type: str="function"
    function: Optional[Dict]

classResponseFormat(BaseModel):
    type: str="json_schema"
    json_schema: Optional[Dict[str, str]]

classSystemMessage(BaseModel):
    role: str="system"
    content: Union[str, List[str]]

classUserMessage(BaseModel):
    role: str="user"
    content: Union[str, List[Union[TextContent, ImageContent, AudioContent]]]

classAssistantMessage(BaseModel):
    role: str="assistant"
    content: Union[str, List[TextContent]] =None
    audio: Optional[Dict[str, str]] =None
    tool_calls: Optional[List[Dict]] =None

classToolMessage(BaseModel):
    role: str="tool"
    content: Union[str, List[str]]
    tool_call_id: str

classChatCompletionRequest(BaseModel):
    context: Optional[Dict] =None
    model: Optional[str] =None
    messages: List[Union[SystemMessage, UserMessage, AssistantMessage, ToolMessage]]
    response_format: Optional[ResponseFormat] =None
    modalities: List[str] =["text"]
    audio: Optional[Dict[str, str]] =None
    tools: Optional[List[Tool]] =None
    tool_choice: Optional[Union[str, ToolChoice]] ="auto"
    parallel_tool_calls: bool=True
    stream: bool=True
    stream_options: Optional[Dict] =None

security =HTTPBearer()

defverify_token(credentials: HTTPAuthorizationCredentials =Depends(security)):
    token =credentials.credentials
    iftoken !=os.getenv("API_TOKEN"):
        logger.warning("Invalid or missing token")
        raiseHTTPException(status_code=403, detail="Invalid or missing token")

@app.post("/chat/completions", dependencies=[Depends(verify_token)])
async defcreate_chat_completion(request: ChatCompletionRequest, req: Request):
    try:
        logger.debug(f"Received request: {request.json()}")
        #TODO Your logic here
        yield"data: [DONE]\n\n"
    exceptasyncio.CancelledError:
        logger.info("Request was cancelled")
        raiseHTTPException(status_code=499, detail="Request was cancelled")
    exceptException as e:
        traceback_str =''.join(traceback.format_tb(e.__traceback__))
        error_message =f"{str(e)}\n{traceback_str}"
        logger.error(error_message)
        raiseHTTPException(status_code=500, detail=error_message)

if__name__ =="__main__":
    importuvicorn
    fromfastapi.security importHTTPBearer, HTTPAuthorizationCredentials
    fromfastapi importDepends

    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Vision Modality

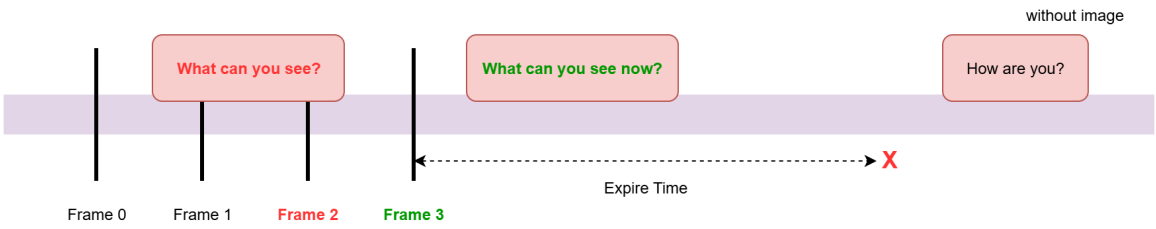
We now support image as a modality, and that need the custom llm to support as well.

The protocol is the same as OpenAI:

```
{
    "role": "user",
    "content": [{
        "type": "image_url",
        "image_url": {
            "url": "jpeg_base64encoded"
        }
    }]
}
```

Agent will record rtc video frame every one second. Each time a new query is to be sent to custom llm, the encoded video frame will be sent along:

- Client can send live video stream, and every query will carry the latest frame received.
- Client can send still image via video stream temporarily when a query of vision modality is need.
- The video frame will expired after 3 seconds.



Transcription

We use datastream to transfer transcription for both user and agent:

- iOS:https://doc.shengwang.cn/api-ref/rtc/ios/API/toc_datastream#callback_irtcengineeventhandler_onstreammessage
- Android:https://doc.shengwang.cn/api-ref/rtc/android/API/toc_datastream#onStreamMessage

It will be sent during the conversation in the same channel:

- is_final: whether the transcription will change later (**Always True**)
- stream_id: rtc int uid for user, 0 for agent and non-0 for user.
- message_id: unique message id
- data_type: “transcribe” as default
- text_ts: timestamp in millisecond
- text: transcription text.

One transcription will be divided into several packets encoded as{message_id}|{id}|{total}|{chunked_data}, and data will be in base64 encoding.

raw

```
{
    "is_final": true,
    "stream_id": 0,
    "message_id": "abcdefgh",
    "data_type": "transcribe",
    "text_ts": 1111111111,
    "text": "Once upon a time, in a small village nestled between rolling hills, there was a
mysterious forest known as the Whispering Woods. The villagers often spoke of the woods in hushed
tones, for it was said that the trees could speak, sharing secrets of the past and visions of the
future. However, few dared to enter, fearing the unknown.One day, a curious girl named Elara decided
to explore the woods. She had always been fascinated by the tales told by the village elders. With a
heart full of courage and a sense of adventure, she set off at dawn, leaving behind the familiar
sounds of her village.As Elara stepped into the Whispering Woods, she was greeted by a gentle breeze
that rustled the leaves above. It felt as if the trees were welcoming her. The deeper she ventured,
the more alive the forest seemed. She could hear faint whispers, like a soft melody carried by the
wind.“Who’s there?” Elara called out, her voice echoing in the stillness. To her surprise, the
whispers grew louder, forming words that danced around her.“Seek the heart of the woods, and you shall
find what you seek,” the trees murmured in unison.Intrigued, Elara followed the sound, her heart
racing with anticipation. After walking for what felt like hours, she came upon a clearing bathed in
sunlight. In the center stood an ancient tree, its trunk wide and gnarled, with branches that reached
towards the sky. It was the oldest tree in the forest, known as the Heartwood.Approaching the
Heartwood, Elara placed her hand on its rough bark. Suddenly, she felt a warmth spreading through her
fingers, and the whispers transformed into a clear voice. “You have shown great courage, young one.
What is it that you seek?”Elara took a deep breath. “I seek knowledge and understanding. I want to
know my purpose in this world.”The Heartwood paused, and Elara felt the air around her shimmer. “Your
purpose is not a single path, but a tapestry woven from your choices, your kindness, and your dreams.
Embrace your journey, and you will find your way.”With that, the whispers faded, and a sense of peace
enveloped Elara. She realized that her journey was just beginning. Thanking the Heartwood, she made
her way back through the woods, feeling lighter and more connected to the world around her.When she
returned to the village, Elara shared her experience with the villagers. Inspired by her bravery,
others began to explore the Whispering Woods, discovering their own paths and purposes.From that day
on, the village thrived, with each person finding their unique place in the world. And the Whispering
Woods remained a sanctuary of wisdom, welcoming all who sought its secrets.And so, the story of Elara
and the Whispering Woods became a cherished tale, reminding everyone that the journey of self-
discovery is filled with wonder and possibility."
}
```


base64 encoded

```
ewoJImlzX2ZpbmFsIjogdHJ1ZSwKCSJzdHJlYW1faWQiOiAwLAoJIm1lc3NhZ2VfaWQiOiAiYWJjZGVmZ2giLAoJImRhdGFfdHlwZSI
6ICJ0cmFuc2NyaWJlIiwKCSJ0ZXh0X3RzIjogMTExMTExMTExMSwKCSJ0ZXh0IjogIk9uY2UgdXBvbiBhIHRpbWUsIGluIGEgc21hbG
wgdmlsbGFnZSBuZXN0bGVkIGJldHdlZW4gcm9sbGluZyBoaWxscywgdGhlcmUgd2FzIGEgbXlzdGVyaW91cyBmb3Jlc3Qga25vd24gY
XMgdGhlIFdoaXNwZXJpbmcgV29vZHMuIFRoZSB2aWxsYWdlcnMgb2Z0ZW4gc3Bva2Ugb2YgdGhlIHdvb2RzIGluIGh1c2hlZCB0b25l
cywgZm9yIGl0IHdhcyBzYWlkIHRoYXQgdGhlIHRyZWVzIGNvdWxkIHNwZWFrLCBzaGFyaW5nIHlvdXIgc2VjcmV0cywgc2hlIHBhc3Qg YW5
kIHZpc2lvbnMgb2YgdGhlIGZ1dHVyZS4gSG93ZXZlciwgZmV3IGRhcmVkIHRvIGVudGVyLCBmZWFyaW5nIHRoZSB1bmtub3duLk9uZS
BkYXksIGEgY3VyaW91cyBnaXJsIG5hbWVkIEVsYXJhIGRlY2lkZWQgdG8gZXhwbG9yZSB0aGUgd29vZHMuIFNoZSBoYWQgYWx3YXlzI
GJlZW4gZmFzY2luYXRlZCBieSB0aGUgdGFsZXMgdG9sZCBieSB0aGUgdmlsbGFnZSBlbGRlcnMuIFdpdGggYSBoZWFydCBmdWxsIG9m
IGNvdXJhZ2UgYW5kIGEgc2Vuc2Ugb2YgYWR2ZW50dXJlLCBzaGUgc2V0IG9mZiBhdCBkYXduLCBsZWF2aW5nIGJlaGluZCB0aGUgZmF
taWxpYXIgc291bmRzIG9mIGhlciB2aWxsYWdlLkFzIEVsYXJhIHN0ZXBwZWQgaW50byB0aGUgV2hpc3BlcmluZyBXb29kcywgc2hlIH
dhcyBncmVldGVkIGJ5IGEgZ2VudGxlIGJyZWV6ZSB0aGF0IHJ1c3RsZWQgdGhlIGxlYXZlcyBhYm92ZS4gSXQgZmVsdCBhcyBpZiB0a
GUgdHJlZXMgd2VyZSB3ZWxjb21pbmcgaGVyLiBUaGUgZGVlcGVyIHNoZSB2ZW50dXJlZCwgdGhlIG1vcmUgYWxpdmUgdGhlciBtYWs
uZC7igJxXaG
/igJlzIHRoZXJlP+KAnSBFbGFyYSBjYWxsZWQgb3V0LCBoZXIgdm9pY2UgZWNob2luZyBpbiB0aGUgc3RpbGxuZXNzLiBUbyBoZXIgc
3VycHJpc2UsIHRoZSB3aGlzcGVycyBncmV3IGxvdWRlciwgZm9ybWluZyB3b3JkcyB0aGF0IGRhbmNlZCBhcm91bmQgaGVyLuKAnFNl
ZWsgdGhlIGhlYXJ0IG9mIHRoZSB3b29kcywgYW5kIHlvdSBzaGFsbCBmaW5kIHdoYXQgeW91IHNlZWss4oCdIHRoZSB0cmVlcyBtdXJ
tdXJlZCBpbiB1bmlzb24uSW50cmlndWVkLCBFbGFyYSBmb2xsb3dlZCB0aGUgc291bmQsIGhlciBuZWFydCByYWNpbmcgd2l0aCBhbn
RpY2lwYXRpb24uIEFmdGVyIHdhbGtpbmcgZm9yIHdoYXQgZmVsdCBsaWtlIGhvdXJzLCBzaGUgY2FtZSB1cG9uIGEgY2xlYXJpbmcgY
mF0aGVkIGluIHN1bmxpZ2h0LiBJbiB0aGUgY2VudGVyIHN0b29kIGFuIGFuY2llbnQgdHJlZSwgaXRzIHRydW5rIHdpZGUgYW5kIGdu
YXJsZWQsIHdpdGggYnJhbmNoZXMgdGhhdCByZWFjaGVkIHRvd2FyZHMgc4gSXQgd2FzIHRoZSBvbGRlc3QgdHJlZSBpbiB
0aGUgZm9yZXN0LCBrbm93biBhcyB0aGUgSGVhcnR3b29kLkFwcHJvYWNoaW5nIHRoZSBIZWFydHdvb2QsIEVsYXJhIHBsYWNlZCBoZX
IgaGFuZCBvbiBpdHMgcm91Z2ggYmFyay4gU3VkZGVubHksIHNwcmVhZGluZyB0aHJvdWdoIGhlciBma
W5nZXJzLCBhbmQgdGhlIHdoaXNwZXJzIHRyYW5zZm9ybWVkIGludG8gYSBjbGVhciB2b2ljZS4g4oCcWW91IGhhdmUgc2hvd24gYmFy
YXQgY291cmFnZSwgeW91bmcgb25lLiBXaGF0IGlzIGl0IHRoYXQgeW91IHNlZWs
/4oCdRWxhcmEgdG9vayBhIGRlZXAgYnJlYXRoLiDigJxJIHNlZWsga25vd2xlZGdlIGFuZCB1bmRlcnN0YW5kaW5nLiBJIHdhbnQgdG
8ga25vdyBteSBwdXJwb3NlIGluIHRoaXMgd29ybGQu4oCdVGhlIEhlYXJ0d29vZCBwYXVzZWQsIGFuZCBFbGFyYSBmZWx0IHRoZSBha
XIgYXJvdW5kIGhlciBzaGltbWVyLiDigJxZb3VyIHB1cnBvc2UgaXMgbm90IGEgc2luZ2xlIHBhdGgsIHJ1IHdv
dmVuIGZyb20geW91ciBjaG9pY2VzLCB5b3VyIGtpbmRuZXNzLCBhbmQgeW91ciBkcmVhbXMuIEVtYnJhY2UgeW91ciBqb3VybmV5LCB
hbmQgeW91IHdpbGwgZmluZCB5b3VyIHdheS7igJ1XaXRoIHRoYXQsIHRoZSB3aGlzcGVycyBmYWRlZCwgYW5kIGEgc2Vuc2Ugb2YgcG
VhY2UgZW52ZWxvcGVkIEVsYXJhLiBTaGUgcmVhbGl6ZWQgdGhhdCBoZXIgam91cm5leSB3YXMganVzdCBiZWdpbmm5lY3RlZCB0byByB
ZCBtb3JlIGNvbm5lY3RlZCB0byB0aGUgd29ybGQgYXJvdW5kIGhlci5XaGVuIHNoZSB3b29kcywgZmVlbGluZyBsaWdodGVyIGFu
hcmEgc2hhcmVkIGhlciBleHBlcmllbmNlIHdpdGggZWJqaCBwZXJzb24gZmluZGluZyB0aGVpciBvd24gcGVyc29uYWxpdHkgYXV0aG
BiZWdhbiB0byByBleHBsb3JlIHRoZSBXaGlzcGVyaW5nIFdvb2RzLCBkaXNjb3lcmluZyB0aGVpciBvd24gcGF0aHMgYW5kIHB1cnBvc
2VzLkZyb20gdGhhdCBkYXkgb24sIHRoZSB2aWxsYWdlIHRocml2ZWQsIHdpdGggZWFjaCBwZXJzb24gZmluZGluZyB0aGVpciB1bmlx
dWUgcGxhY2UgaW4gdGhlIHdvcmxkLiBBBbmQgdGhlIFdoaXNwZXJpbmcgV29vZHMgcmVtYWluZWQgYSBzYW5jdHVhcnkgb2Ygd2lzZG9
tLCB3ZWxjb21pbmcgYWxsIHdobyBzb3VnaHQgaXRzIHNlY3JldHMuQW5kIHNvLCB0aGUgc3Rvcnkgb2YgRWxhcmEgYW5kIHRoZSBXaG
lzcGVyaW5nIFdvb2RzIGJlY2FtZSBhIGNoZXJpc2hlZCB0YWxlLCByZW1pbmRpbmcgZXZlcnlvbmUgdGhhdCB0aGUgam91cm5leSBvZ
iBzZWxmLWRpc2NvdmVyeSBpcyBmaWxsZWQgd2l0aCB3b25kZXIgYW5kIBvc3NpYmlsaXR5LiIKfQ==
```

after divided in 5 packets

```
abcdefgh|0|5|ewoJImlzX2Zpb...
abcdefgh|1|5|ZWF2aW5nI...
abcdefgh|2|5|BoZWFydCByYW...
abcdefgh|3|5|G8ga25vdyBt...
abcdefgh|4|5|d24gcGF0aHMgY...
```

Appendix

Voice ID

- en-US-AvaMultilingualNeural
- en-US-AndrewMultilingualNeural
- en-US-EmmaMultilingualNeural
- en-US-BrianMultilingualNeural
- en-US-AvaNeural
- en-US-AndrewNeural
- en-US-EmmaNeural
- en-US-BrianNeural
- en-US-JennyNeural
- en-US-GuyNeural
- en-US-AriaNeural
- en-US-DavisNeural
- en-US-JaneNeural
- en-US-JasonNeural
- en-US-SaraNeural
- en-US-TonyNeural
- en-US-NancyNeural

- en-US-AmberNeural
- en-US-AnaNeural
- en-US-AshleyNeural
- en-US-BrandonNeural
- en-US-ChristopherNeural
- en-US-CoraNeural
- en-US-ElizabethNeural
- en-US-EricNeural
- en-US-JacobNeural
- en-US-JennyMultilingualNeural4
- en-US-MichelleNeural
- en-US-MonicaNeural
- en-US-RogerNeural

ErrorCode

Code	Description
400	Invalid request parameters
403	Unauthorized access
409	Agent conflict
422	Access limit exceeded
502	Gateway error
503	Agent startup error
504	Timeout error