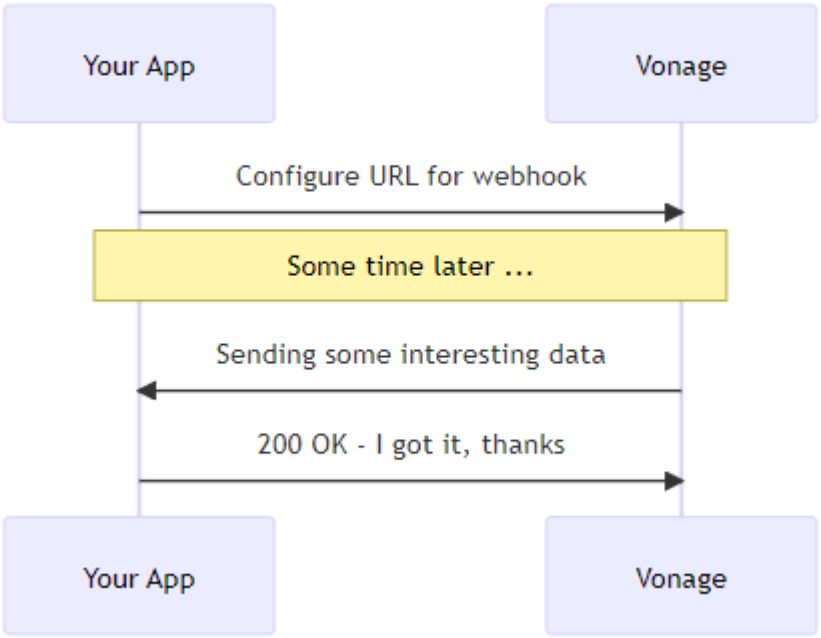
Vonage Voice API

The Vonage Voice API allows you to connect people around the world and automate voice interactions that deliver a frictionless extension of your brand experience using AI technologies.

- Text to Speech with over 50+ languages
- Create IVRs and Voice Bots
- Speech to text and WebSockets
- Embed calls in web and mobile apps
- Web & Mobile (iOS, Android) SDKs
- Record and store inbound or outbound calls
- Send text-to-speech messages in 40 languages with different genders and accents
- Create conference calls

Voice API Webhooks



Webhooks provide a convenient mechanism for Vonage to send information to your application for events such as an incoming call or message, or a change in call status.

Webhooks

- Answer Webhook
- Event Webhook
- Fallback URL

Answer Webhook

- Answer webhook is sent when a call is answered. This is for both incoming and outgoing calls.
- When an incoming call is answered, an HTTP request is sent to the answer_url.

Answer webhook data fields

Fields	**Description**
to	The number that answered the call.
from	The number that called **to.** This could be a landline or mobile number
From_user	The username that called **to** only if the call was made using the client SDK.
uuid	Unique identifier for this call.
Conversation_uuid	Unique identifier for this conversation
Region_url	Regional API endpoint which should be used to control the call with **REST API**.
Custom_data	Custom data object, optionally passed as parameter.

Event webhook

- Event webhook is sent for all the events that occur during a call. Your application can log, react to or ignore each event type.
- HTTP requests will arrive at the event webhook endpoint when there is any status change for a call.
- By default the incoming requests are POST requests with a JSON body.

Event webhook data fields

Fields	Descriptions
from	The number the call came from
to	The number the call was made to
uuid	Unique identifier this call
conversation_uuid	Unique identifier this conversation
status	Call status
direction	Call direction
timestamp	Timestamp (ISO format)

Status Types

Started

Ringing

Answered

Busy

Cancelled

Unanswered

Disconnect

Rejection

Failed

Timeout

Human / machine

Completed

Record

Input

Transfer

Direction Type

Inbound

Outbound

Fallback URL

Fallback URL is used when either the Answer or Event webhook fails or returns an HTTP error status.

example

```
{
  "reason": "Connection closed.",
  "original_request": {
    "url": "https://api.example.com/webhooks/event",
    "type": "event"
  }
}
```

NCCO - Nexmo Call Control Objects

A Call Control Object (NCCO) is represented by a JSON array. You can use it to control the flow of a Voice API call.

The Call event model is asynchronous. When a Call is placed to your number, Vonage makes a synchronous request to the webhook endpoint you set as the answer_url for your number and retrieves the NCCO object that controls the Call.

NCCO instruction are:

- Action - something to be done in the Call.
- Option - how to customize an *action*.
- Type - describes an *option*. For example, type=phone for an endpoint option.

actions you can use in an NCCO are:

- record - all or part of a call
- conversation - create a standard or hosted conversation
- connect - connect to a connectable endpoint such as a phone number or Vonage Business Cloud extension
- talk - send synthesized speech to a conversation
- stream - send audio files to a conversation
- input - collect digits from the person you are calling, then process them

Creating a custom call or conversation for each user

When you make an outbound call or accept an inbound call, Vonage makes a request to your webhook endpoint at answer_url and retrieves your NCCO. This request contains the following parameters:

Name	Description
to	The endpoint being called.
from	The endpoint you are calling from.
conversation_uuid	Unique ID for this conversation
uuid	Unique ID for this call

Code examples show how to provide the NCCO that controls your call or conversation

```
from flask import Flask, request, jsonify

app = Flask(__name__)

HOST = "localhost"
PORT = 3000

@app.route("/webhooks/answer")
def answer_call():
    call_from = request.args['from']

    # Dynamically build NCCO based on source phone
    if call_from == "447700900000":
        ncco = [{
                "action": "talk",
                "text": "Hi John, we will be with you shortly."
            }]
    elif call_from == "447700900001":
        ncco = [{
                "action": "talk",
                "text": "Hi Jane, we will be with you shortly."
            }]
    else:
            ncco = [{
                "action": "talk",
                "text": "Hello, sorry, we do not recognize your number."
            }]
    return jsonify(ncco)

if __name__ == 'main':
    app.run(host=HOST, port=PORT
```

WebSockets

WebSockets is a computer communications protocol that enables two-way communication over a single, persistent TCP connection without the overhead of the HTTP request/response model.

Using Vonage's Voice API, you can connect phone calls to WebSocket endpoints. This means that any application that hosts a WebSocket server can be a participant in a Vonage voice conversation. It can receive raw audio from and play audio into the call in real time.

Automating calls with bots to perform tasks such as food ordering or requesting information from field experts.

The endpoint is addressed via a uri parameter which should be a standard websocket URL, starting with either ws:// for plain HTTP or wss:// for TLS enabled servers.

WebSockets allow you to connect phone calls to any AI bot engine of your choice.

Working with WebSockets

when establishing the WebSocket connection.

- Return an NCCO instructing Vonage to connect to your WebSocket endpoint
- Accept this WebSocket connection
- Handle JSON text-based protocol messages
- Handle mixed call audio binary messages

Connecting to a WebSocket

- Vonage to connect to a WebSocket your application server must return an NCCO when requested from your Vonage Application's answer_url
- NCCO must contain a connect action with an endpoint.type of websocket

Example

```
[
    {
       "action": "connect",
       "endpoint": [
           {
                "type": "websocket",
                "uri": "wss://example.com/socket",
                "content-type": "audio/l16;rate=16000",
                "headers": {
                    "language": "en-GB",
                    "caller-id": "447700900123"
                }
           }
       ]
```

```
    }
]
```

The specific data fields for webhooks

Fields	Descriptions
uri	Endpoints of your websocket server that vonage will connect to
Content-type	String representing the audio sampling rate, **audio/l16;rate=16000** or **audio/l16;rate=8000** and **audio at 8kHz**.
headers	additional optional properties to send to your Websocket server

Handling incoming WebSocket messages

The initial message sent on an established WebSocket connection will be text-based and contain a JSON payload, it will have the event field set to websocket:connected and detail the audio format in content-type, along with any other metadata that you have put in the headers property of the WebSocket endpoint in your NCCO connect action.

example

```
{
    "event":"websocket:connected",
    "content-type":"audio/l16;rate=16000",
    "prop1": "value1",
    "prop2": "value2"
}
```

Binary audio messages

Messages that are binary represent the audio of the call. The audio codec presently supported on the WebSocket interface is Linear PCM 16-bit, with either a 8kHz or a 16kHz sample rate, and a 20ms frame size.

To choose the sampling rate set the Content-Type property to audio/l16;rate=16000 or audio/l16;rate=8000 depending on if you need the data at 16kHz or 8kHz.

Sampling rate	Number of samples in 20ms	Bytes per message
8000	160	160 * 2 = 320
16000	320	320 * 2 = 640

Call Flow

Inbound calls are made to the Vonage platform by one of the following methods:

- to a Vonage number from a regular phone,
- from a client application using the Client SDK.

Outbound calls are calls made from the Vonage platform to

- a regular phone number,
- client application, or
- WebSocket server.

Outbound calls are usually initiated in response to a request made via the REST API to create a new call.

Two types of call flow

- **Scripted Call**: when the flow is determined by a sequence of question-answer steps (actions);
- **Live Conversation**: which connects two or more participants in a conversation.

Scripted Call:

inbound and outbound calls initially follow the same call flow once answered. This call flow is controlled by an NCCO.

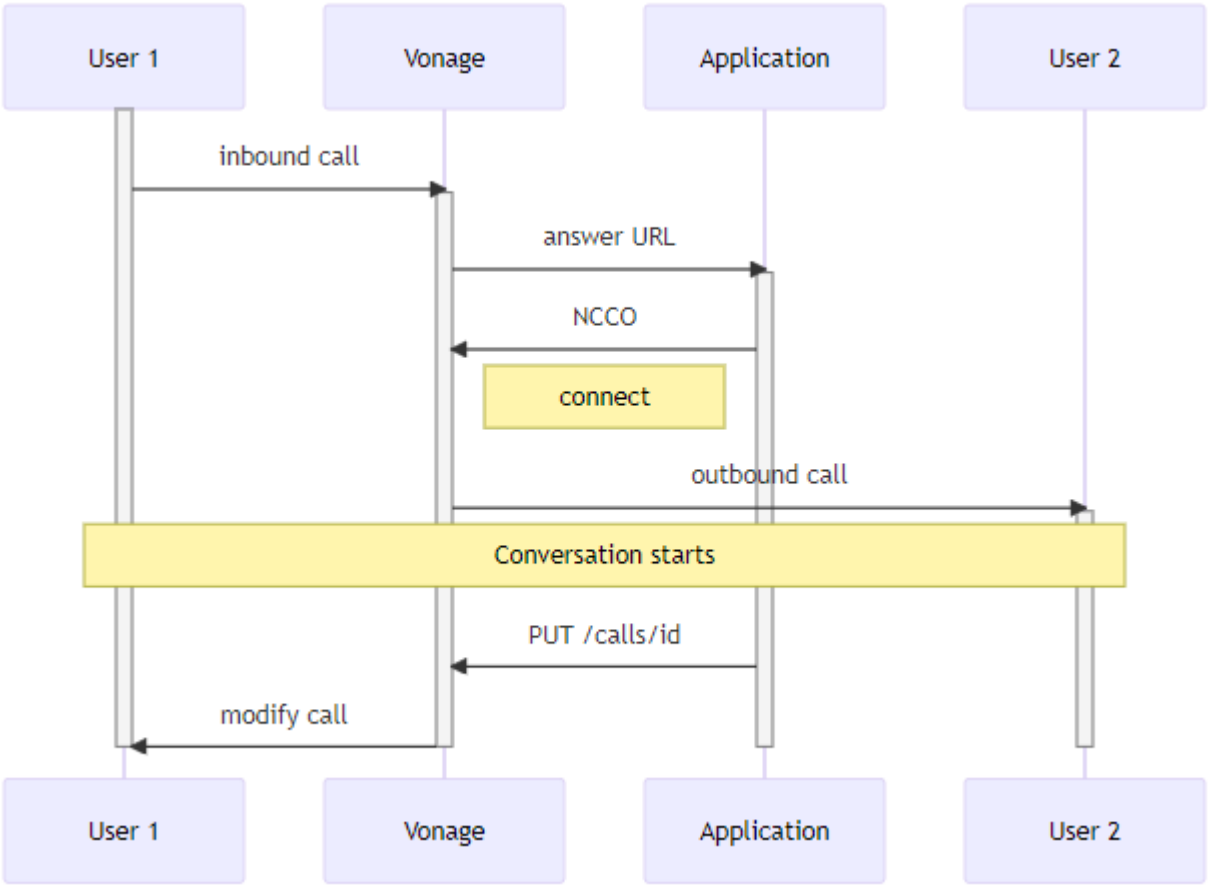
- inbound calls, the `answer_url` is configured in your Voice Application.
- outbound calls, you provide an `answer_url` in the API request that creates the call.

Live Conversation:

A **private voice communication** use case, you want to connect two or more participants to establish a live conversation.

Each call, inbound or outbound, is automatically added to the new conversation behind the scenes.

- Create a new outbound call with the `connect` action - it will be automatically joined to the same conversation;
- Move the call to an existing (or new) named conversation with the `conversation` action.



Since any type of voice endpoint might be used in the connect action, **the second member is not necessarily a human**: it might be a **voice bot** talking to the user using the media passed through the **WebSocket connection**.

Numbers Format

Numbers are a key concept to understand when working with the Vonage Voice API.

Formatting

Vonage Voice API all numbers are in E.164 format.

- Omit both a leading `+` and the international access code such as `00` or `001`.
- Contain no special characters, such as a space, `()` or `-`

For example, a US number would have the format `14155550101`. A UK number would have the format `447700900123`.

Outgoing CallerID

When making an outbound call from Vonage the CallerID, `from` value needs to be a Vonage Number associated with your account.

Incoming Call Numbers

Vonage offers for rental incoming numbers located in many countries around the world. In some countries the numbers may be enabled for SMS or Voice only, or in others they will support both

Vonage can also provide numbers in both 'landline' and 'mobile' ranges for many countries.

You can link multiple incoming numbers to the same application and the number that was called will be passed to your `answer_url` in the `to` parameter.

Endpoints

Endpoints is a term used with the Vonage Voice API to describe the different destinations a call could be connected to.

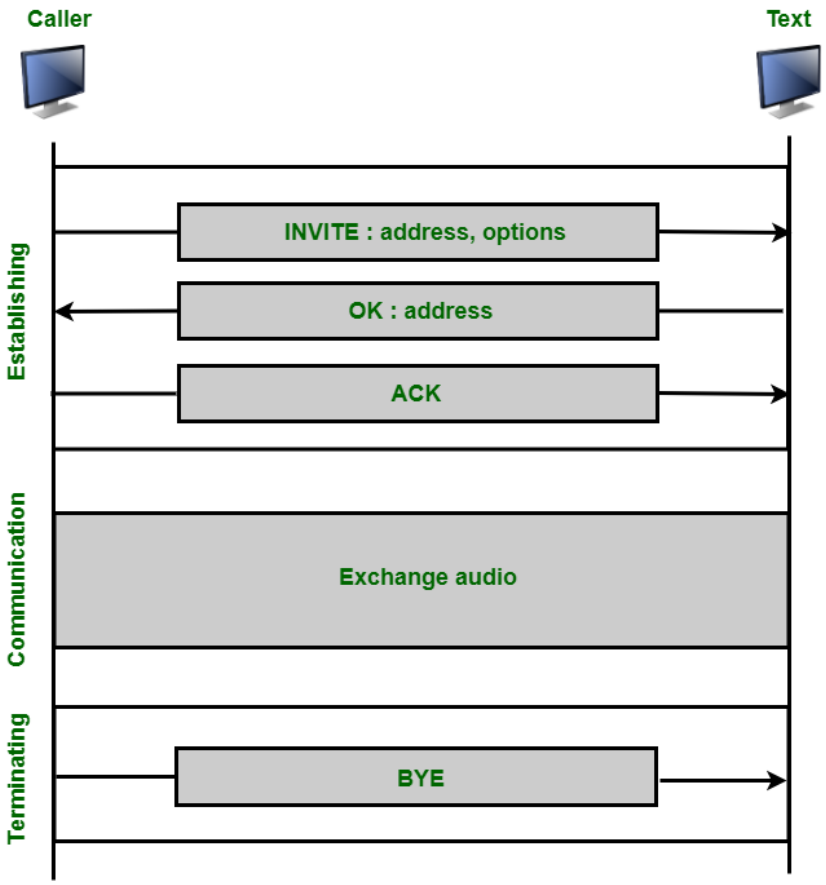
The most common type of endpoint used is phone, which is for making phone calls to regular phone numbers anywhere in the world.

Phone

The phone endpoint takes a required value of number which is the number to be called in E.164 format. E.164 format is the full international format without any leading zeros or + signs

Session Initiation Protocol (SIP)

The Session Initiation Protocol (SIP) is a signaling protocol used for initiating, maintaining, and terminating communication sessions that include voice, video and messaging applications. SIP is used in Internet telephony, in private IP telephone systems, as well as mobile phone calling over LTE (VoLTE).



SIP is used to connect a call to your own SIP system such as an office PBX or other telephony service using the standards laid down in RFC3261.

The uri should be constructed as a SIP URL in the format sip:user@example.com by default Vonage will connect to port 5060 unless another port is specified in the URI. You can specify TLS for the SIP transport by adding ;transport=tls to the end of your URI.

Media will be sent via UDP on all ports.

CallerID

For both phone and sip endpoint types, the from field must be a Vonage Number associated with your account in E.164 format. This will then be used as the caller ID on the receiving phone. For SIP endpoints it will take the format number@sip.nexmo.com

Text to Speech

Vonage uses text-to-speech engines to allow you to play machine generated speech to your users. This can either be done via an NCCO with the use of the `talk` action, or by making a PUT request to an in-progress call.

Example

```
[
  {
    "action": "talk",
    "text": "Thank you for calling. Please leave your message after the tone."
  }
]
```

Locale

You can set the language code (BCP-47) with a `language` parameter in the `talk` command, if you do not specify a language then Vonage will default to an `en-US` voice.

The `style` parameter maps to features such as vocal range, timbre and tessitura of the selected voice.

Example

```
[
  {
    "action": "talk",
    "text": "Obrigado pela sua chamada. Por favor, deixe sua mensagem após o
sinal.",
    "language": "pt-PT",
    "style": 6
  }
]
```

Premium Voices

Some voice styles come with a premium alternative, which through the use of AI, have a more natural sound.

add the `premium` option in your NCCO:

```
[
  {
    "action": "talk",
    "text": "Obrigado pela sua chamada. Por favor, deixe sua mensagem após o
sinal.",
    "language": "pt-PT",
    "style": 6,
    "premium": true
  }
]
```

Customizing Spoken Text

You can control how the Vonage Voice API plays machine-generated text to your users by using a subset of the tags defined in the **Speech Synthesis Markup Language (SSML)** specification.

This **XML-based markup** enables you to mix multiple languages, provide pronunciation hints for specific words and numbers and control the speed, volume and pitch of synthesized text.

In an **NCCO talk action**, you can send SSML tags as part of the text string.

you must surround the entire string in `<speak></speak>` tags to tell Vonage that the string includes SSML.

Example

```
[
    {
      "action": "talk",
      "text": "<speak><p>Hello.</p><p>How are you?</p></speak>"
    }
]
```

SSML tags

Breaks

The break tag allows you to add pauses to text. The duration of the pause can be specified either using a strength duration or as a time seconds or milliseconds.

```
<speak>My name is <break time='1s' />Slim Shady.</speak>
```

Valid strength values include:

none or **x-weak** (which removes a pause which might otherwise exist after a full stop)

weak or **medium** (equivalent to a comma)

strong or **x-strong** (equivalent to a paragraph break)

```
<speak>
To be <break strength='weak' />
or not to be <break strength='weak' />
that is the question.
</speak>
```

Emphasizing

Emphasizing words changes the speaking rate and volume. More emphasis makes the text spoken louder and slower. Less emphasis makes it quieter and faster.

Emphasis tag is not available for Premium TTS voices.

To specify the degree of emphasis, use the level attribute.

Valid level values include:

strong: Increases the volume and slows the speaking rate so that the speech is louder and slower.

moderate: Increases the volume and slows the speaking rate, but less than strong. moderate is the default.

reduced: Decreases the volume and speeds up the speaking rate. Speech is softer and faster.

```
<speak>
<emphasis level="moderate">This is an important announcement</emphasis>
</speak>
```

Language

- The **lang** tag allows you to specify another language for a specific word, phrase, or sentence.
- It might be useful for better pronunciation of foreign words.
- The language tag should contain both the language tag and country code (e.g. pt-BR for Brazilian Portuguese, en-GB for British English)

```
<speak><lang xml:lang='it-IT'>Buongiorno</lang></speak>
```

Not all the voice styles support lang tag.

Phonemes

The phoneme tag allows you to send an International **Phonetic Alphabet (IPA)** representation of a word.

you need to specify both an alphabet (either ipa or x-sampa) and the ph attribute containing the phonetic symbols.

```
<speak>
    <phoneme alphabet='ipa' ph='təˈmæto:'>Tomato</phoneme> or
    <phoneme alphabet='ipa' ph='təˈmeɪtoʊ'>tomato</phoneme>.
    Two nations separated by a common language.
</speak>
```

Prosody

Prosody tag allows you to set the **pitch, rate and volume of the text**.

The volume attribute can be set to the following values: **default, silent, x-soft, soft, medium, loud and x-loud.**

The rate attribute changes the **speed of speech**. Acceptable values include: **x-slow, slow, medium, fast and x-fast.**

The **pitch attribute** changes the **pitch of the voice**.

When using Premium TTS voices, the pitch attribute is unsupported.

```
<speak>
I am <prosody volume='loud'>loud and proud</prosody>,
<prosody rate='fast'>quick as a cricket</prosody>
and can <prosody pitch='x-low'>change my pitch</prosody>.
</speak>
```

Say As

The say-as tag allows you to provide instructions for how **particular words and numbers are spoken**.

Many of these features are **automatically detected in speech by the TTS engine**

say-as command allows you to mark them specifically.

The **say-as tag** has a required attribute**: interpret-as**.

Value of interpret-as

1. **character**: Spells each letter out
2. **cardinal**: Pronounces the value as a number.
3. **ordinal**: Pronounces the number as an ordinal.
4. **digits**: Reads the specified numbers out as digits.
5. **fraction**: Reads the numbers out as a fraction.
6. **unit**: Reads the specified number out as a unit.
7. **date**: Specify how to pronounce dates.
8. **time**: Pronounces time durations in minutes and seconds.
9. **expletive**: Replaces the content with a "bleep" to censor expletives.
10. **telephone**: Reads out a telephone number with appropriate breaks.
11. **address**: Reads out a street address with appropriate breaks.

Example

```
<speak>
Your number is <say-as interpret-as='cardinal'>10</say-as>.
You are <say-as interpret-as='ordinal'>10</say-as> in line.
The digits for ten are <say-as interpret-as='digits'>10</say-as>.
</speak>
```

Date formatting

Dates can be formatted in the following ways:

format

1. **mdy**: month-date-year (e.g. "3/10/2019")
2. **dmy**: day-month-year (e.g. "10/3/2019")
3. **ymd**: year-month-day (e.g. "2019/3/10")
4. **md**: month-day (e.g. "3/10")
5. **dm**: day-month (e.g. "10/3")
6. **ym**: year-month (e.g. "2019/3")
7. **my**: month-year (e.g. "3/2019")
8. **d**: day (e.g. "10")
9. **m**: month (e.g. "3")
10. **y**: year (e.g. "2019")
11. **yyyymmdd**: year-month-day

Sentences and paragraphs

Sentences

You can wrap sentences in the **s tag**. This is equivalent to putting a full stop at the end of the sentence.

```
<speak>
<s>Thank you Mario</s>
<s>But our princess is in another castle</s>
</speak>
```

Paragraphs

The **p tag** allows you to specify paragraphs in your speech.

```
<speak>
<p>Hello.</p>
<p>How are you?</p>
</speak>
```

Substitution

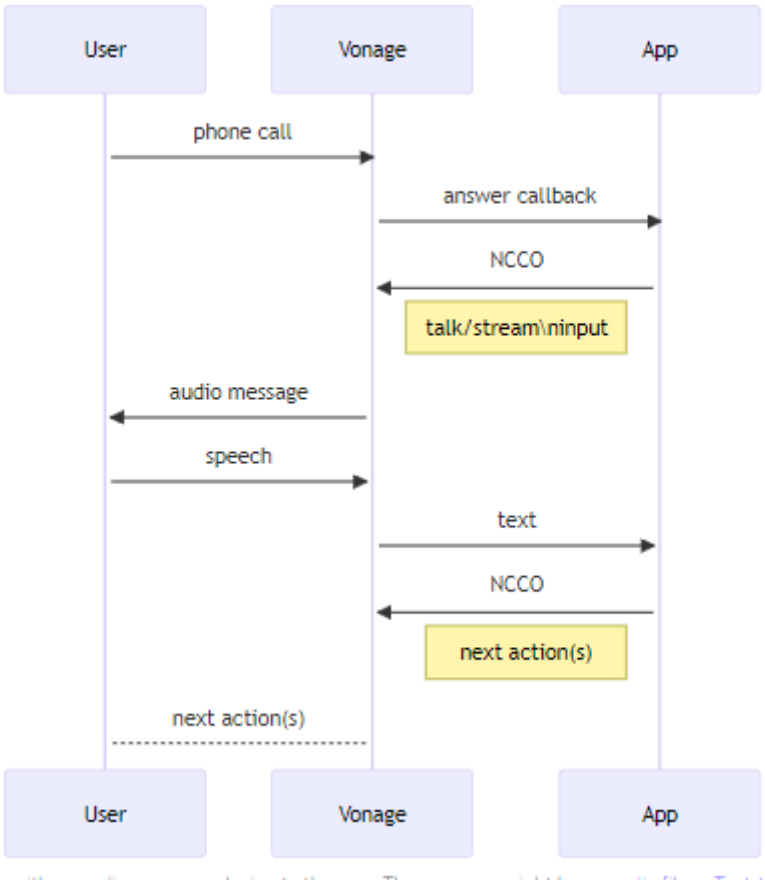
The **sub tag** allows you to provide a substitute pronunciation. The contents of the alias attribute will be read instead.

```
<speak>
Welcome to the <sub alias="United States">US</sub>.
</speak>
```

Speech to Text

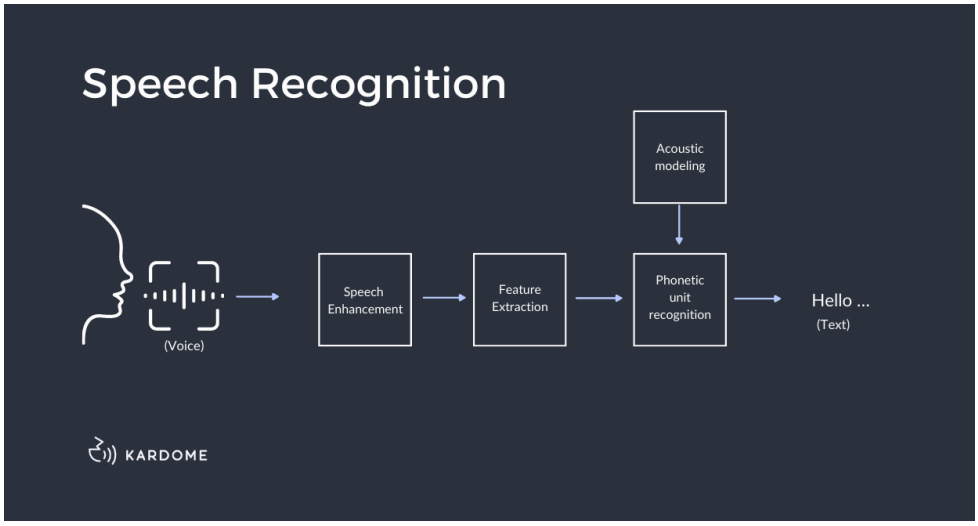
Automatic Speech Recognition (ASR) enables apps to support voice input for such use cases as IVR, identification and different **kinds of voice bots/assistants**. Using this feature, the app gets transcribed **user speech (in the text form)**

How it works



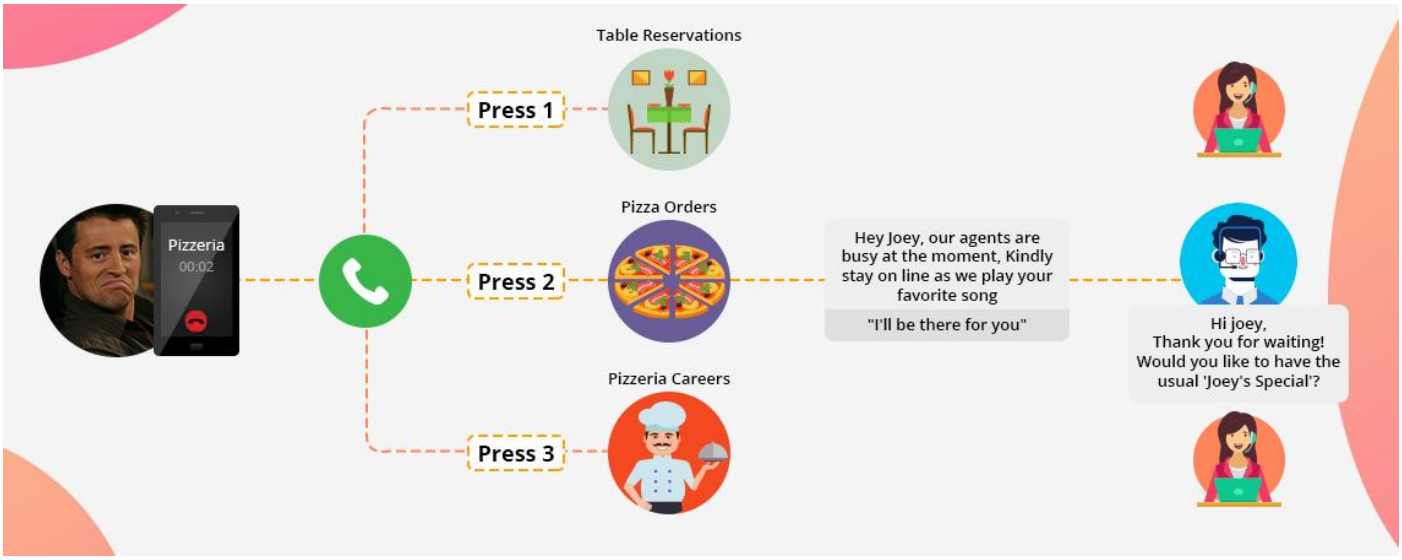
ASR

ASR, is the use of Machine Learning or Artificial Intelligence (AI) technology to process human speech into readable text.



IVR

Interactive Voice Response (IVR) is an automated phone system technology that allows incoming callers to access information via a voice response system of pre recorded messages without having to speak to an agent, as well as to utilize menu options via touch tone keypad selection or speech recognition.



DTMF Tones

Overview

Dual Tone Multi Frequency (DTMF), is a form of signalling used by phone systems to transmit the digits 0-9 and the * and # characters. Typically a caller presses these buttons on their telephone keypad and the phone then generates a tone made up of two frequencies played simultaneously (hence Dual Tone).

This is used to implement an Interactive Voice Response (IVR) system, or to enter information like a PIN number or conference call pin.

The Vonage Voice API supports both collecting information from callers using the input action in an NCCO as well as sending DTMF tones within a call.

Collecting Input

You can collect input from your caller by using the input action within your NCCO.

Once the action is complete, Vonage will send a webhook to your event_url containing the keys that were pressed.

```
[
    {
      "action": "talk",
      "text": "Please enter a digit",
      "bargeIn": true
    },
    {
      "eventUrl": [
        "https://api.example.com/callbacks/events"
      ],
      "action": "input",
      "type": [ "dtmf" ],
      "dtmf": {
        "maxDigits": 1,
        "submitOnHash": true,
        "timeOut": 5
      }
    }
  ]
```

Sending DTMF

There are two ways to send DTMF tones to a call:

- For an outbound call made either via create call endpoint, or via a connect action, you can set the dtmfAnswer parameter within the phone endpoint. This means that when the call is answered, Vonage will automatically send the defined string of tones.

- You can also send DTMF digits to a call at any time by making a PUT request to the REST API, specifying a string of digits

You can use digits 0-9, *, and #. A p indicates a pause of 500ms if you need to add a delay in sending the digits.

Call Recording and Transcription

Overview

The Vonage Voice API offers the ability to record call audio in several ways. You can:

1. Record a call between two people in a passive 'monitor' manner.
2. Record audio from a single caller when they are prompted. For example, in a voicemail system.
3. Enable recording for a named conversation (using the conversation action).

To record a conversation you can use the record action in an NCCO.

Once the record action ends, Vonage will send a webhook to the eventUrl that you specified when configuring the record action.

After your recording is complete, it is stored by Vonage for 30 days before being automatically deleted

Synchronous recording

A record action will complete when either the endOnSilence timer has been reached, or the endOnKey key is sent or the timeout value is reached. At this point the recording will be ended and a record event will be sent to your event_url before the next action is executed.

Asynchronous recording

If none of endOnSilence, endOnKey or timeout is set, then the record will work in an asynchronous manner and will instantly continue on to the next action while still recording the call. The recording will only end and send the relevant event when the call is ended.

Split recording

When recording a call, you can enable split recording which will result in the recording being a stereo file with one channel having the audio sent from the caller and another channel being the audio heard by the caller.

Multi channel recording

When recording a call, you can enable multichannel recording which allows up to 32 call legs to be recorded separately. One file with the number of channels set will be returned.

File formats

- Vonage supports recording in MP3, OGG or WAV format, the default is MP3 (or WAV for recording more than 2 channels).
- MP3 files are recorded with a 16-bit depth and a 16kHz sample rate. They are encoded with a constant bit rate of 32 Kbps.
- WAV files are recorded with a 16-bit depth and a 16kHz sample rate.

Transcription

If the transcription option is set, the recording will be transcribed using the default value for language, en-US:

```
[
    {
        "action": "record",
        "eventUrl":["https://example.com/recording"],
        "transcription": {}
    }
]
```

Masked Calling

This use case shows you how to implement the idea described in Private Voice Communication use case.

using virtual numbers to hide the real phone numbers of the participants.

Sometimes you want two users to be able to call each other without revealing their private phone numbers.

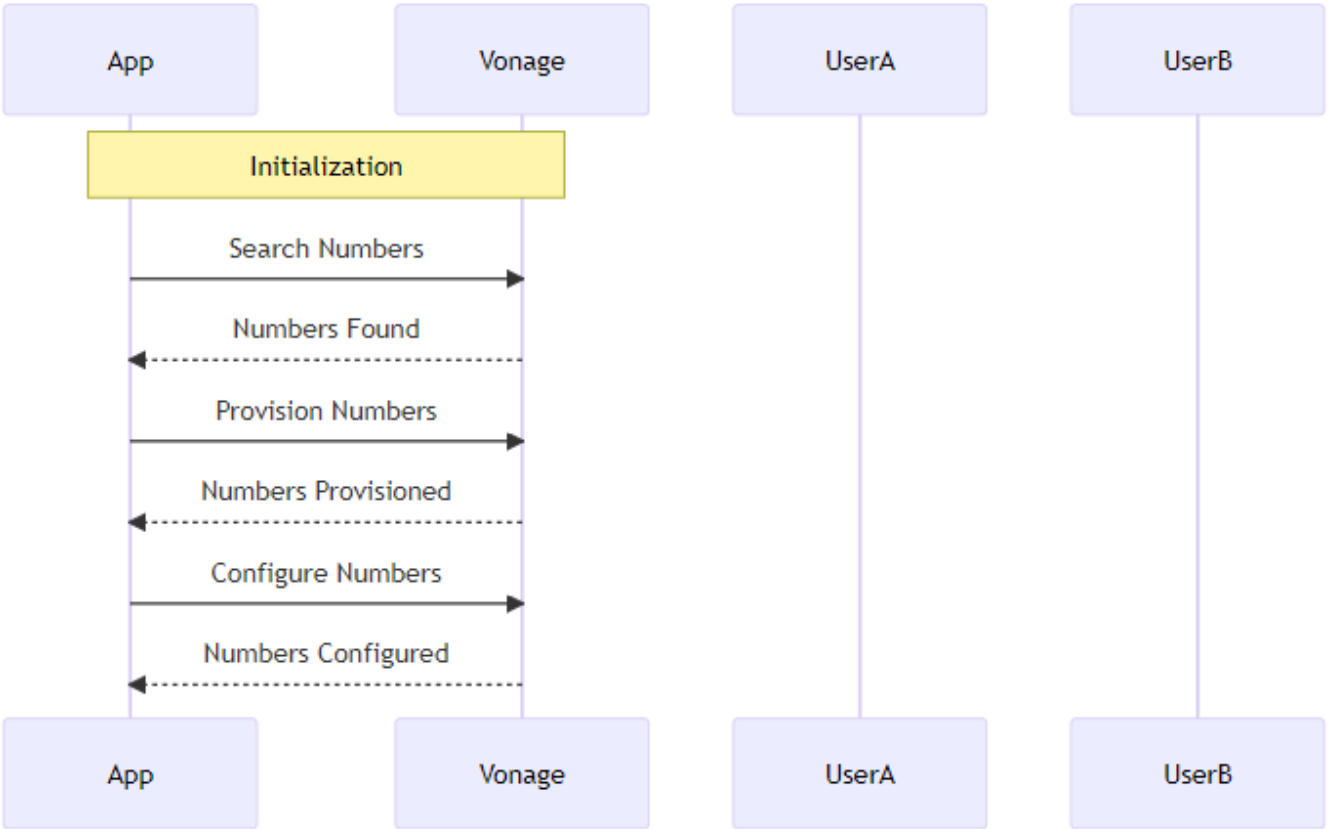
Using Vonage's APIs, you can provide each participant in a call with a temporary number that masks their real number.

Each caller sees only the temporary number for the duration of the call.

Provision virtual numbers

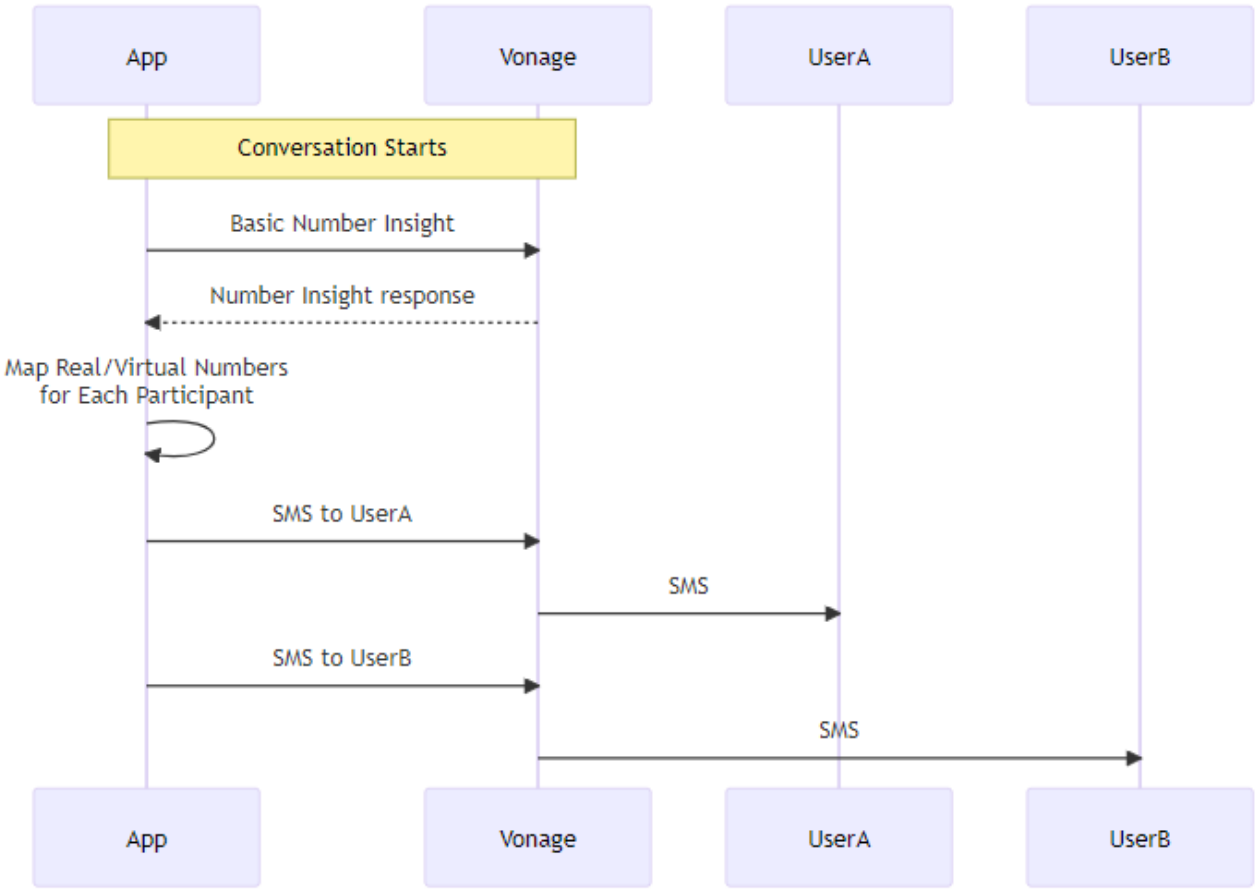
Virtual numbers are used to hide real phone numbers from your application users.

The following workflow diagram shows the process for provisioning and configuring a virtual number:



Create a call

The workflow to create a call is:



Validates the phone numbers

```
/**
 * Create a new tracked conversation so there is a real/virtual mapping of
numbers.
 */
 VoiceProxy.prototype.createConversation = function(userANumber, userBNumber,
cb) {
  this.checkNumbers(userANumber, userBNumber)
    .then(this.saveConversation.bind(this))
    .then(this.sendSMS.bind(this))
    .then(function(conversation) {
      cb(null, conversation);
    })
    .catch(function(err) {
      cb(err);
    });
};
```

Validate the phone numbers

When your application users supply their phone numbers use Number Insight to ensure that they are valid. You can also see which country the phone numbers are registered in:

```
    /**
 * Ensure the given numbers are valid and which country they are associated
with.
 */
VoiceProxy.prototype.checkNumbers = function(userANumber, userBNumber) {
  const niGetPromise = (number) => new Promise ((resolve) => {
    this.nexmo.numberInsight.get(number, (error, result) => {
      if(error) {
        console.error('error',error);
      }
      else {
        return resolve(result);
      }
    })
  });

  const userAGet = niGetPromise({level: 'basic', number: userANumber});
  const userBGet = niGetPromise({level: 'basic', number: userBNumber});

  return Promise.all([userAGet, userBGet]);
};
```

Map phone numbers to real numbers

Once you are sure that the phone numbers are valid, map each real number to a virtual number and save the call:

```
/**
 * Store the conversation information.
 */
 VoiceProxy.prototype.saveConversation = function(results) {
  let userAResult = results[0];
  let userANumber = {
    msisdn: userAResult.international_format_number,
    country: userAResult.country_code
  };

  let userBResult = results[1];
  let userBNumber = {
    msisdn: userBResult.international_format_number,
    country: userBResult.country_code
  };

  // Create conversation object - for demo purposes:
  // - Use first indexed LVN for user A
```

```
  // - Use second indexed LVN for user B
  let conversation = {
    userA: {
      realNumber: userANumber,
      virtualNumber: this.provisionedNumbers[0]
    },
    userB: {
      realNumber: userBNumber,
      virtualNumber: this.provisionedNumbers[1]
    }
  };

  this.conversations.push(conversation);

  return conversation;
};
```

Send a confirmation SMS

In a private communication system, when one user contacts another, the caller calls a virtual number from their phone.

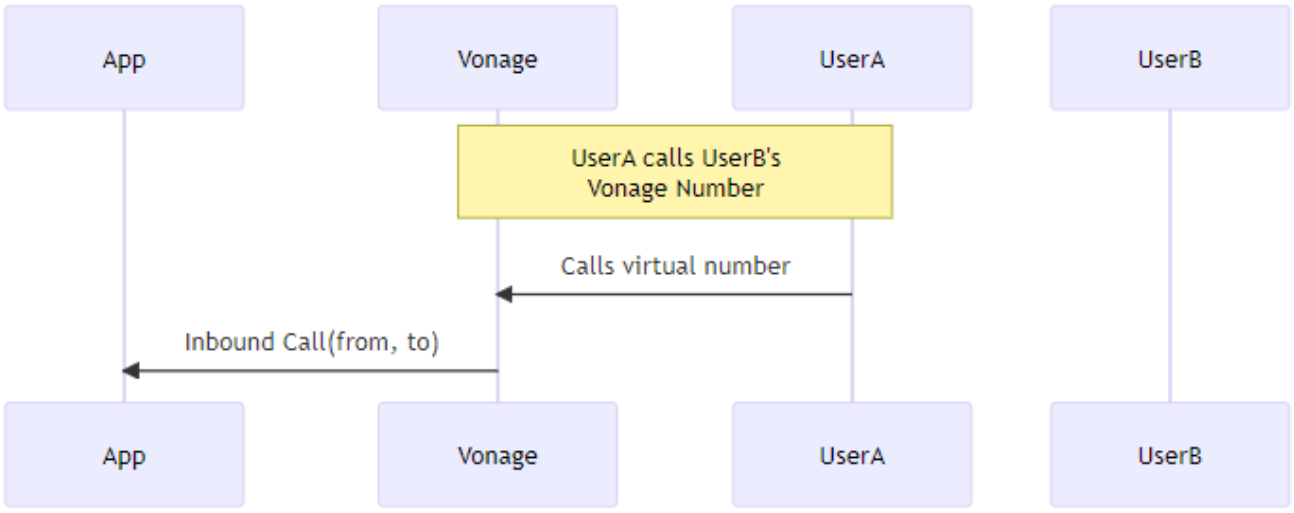
Send an SMS to notify each conversation participant of the virtual number they need to call:

```
/**
 * Send an SMS to each conversation participant so they know each other's
 * virtual number and can call either other via the proxy.
 */
 VoiceProxy.prototype.sendSMS = function(conversation) {
  // Send UserA conversation information
  // From the UserB virtual number
  // To the UserA real number
  this.nexmo.message.sendSms(conversation.userB.virtualNumber.msisdn,
                             conversation.userA.realNumber.msisdn,
                             'Call this number to talk to UserB');

  // Send UserB conversation information
  // From the UserA virtual number
  // To the UserB real number
  this.nexmo.message.sendSms(conversation.userA.virtualNumber.msisdn,
                             conversation.userB.realNumber.msisdn,
                             'Call this number to talk to UserB');

  return conversation;
};
```

The users cannot SMS each other. To enable this functionality you need to setup Private SMS communication.

Handle inbound calls

When Vonage receives an inbound call to your virtual number it makes a request to the webhook endpoint you set when you created a Voice application:

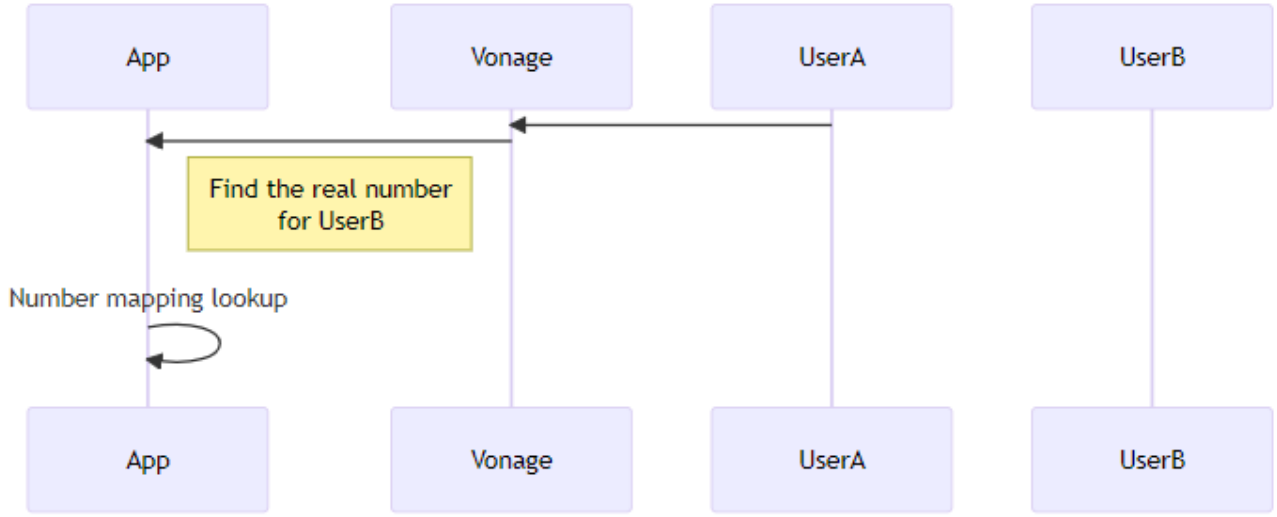


Extract **to** and **from** from the inbound webhook and pass them on to the voice proxy business logic:

```
app.get('/proxy-call', function(req, res) {
  const from = req.query.from;
  const to = req.query.to;

  const ncco = voiceProxy.getProxyNCCO(from, to);
  res.json(ncco);
});
```

Reverse map real phone numbers to virtual numbers

Now you know the phone number making the call and the virtual number of the recipient, reverse map the inbound virtual number to the outbound real phone number:



The call direction can be identified as:

The **from** number is **UserA** real number and the **to** number is **UserB** Vonage number

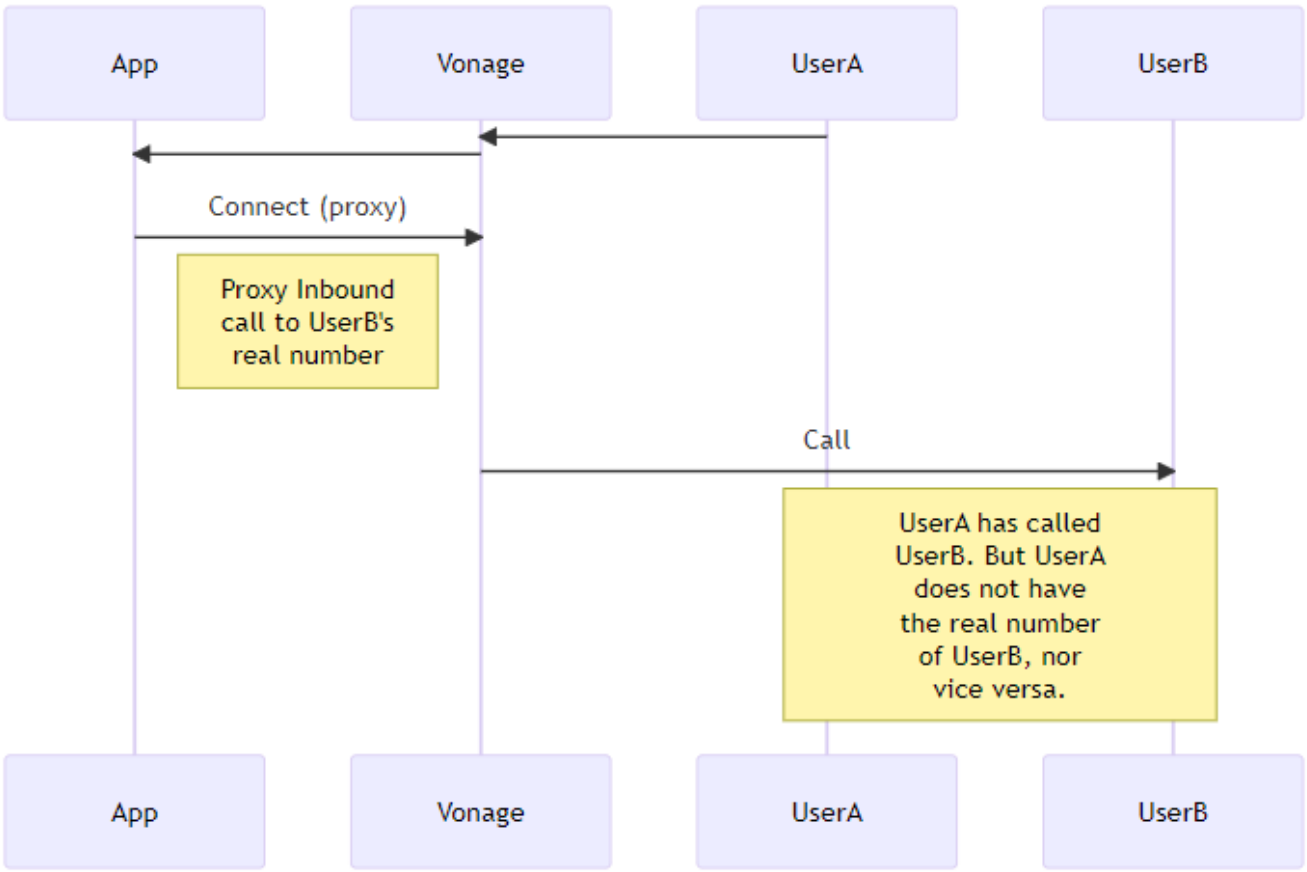
The **from** number is **UserB** real number and the **to** number is **UserA** Vonage number

```
const fromUserAToUserB = function(from, to, conversation) {
  return (from === conversation.userA.realNumber.msisdn &&
          to === conversation.userB.virtualNumber.msisdn);
};
const fromUserBToUserA = function(from, to, conversation) {
  return (from === conversation.userB.realNumber.msisdn &&
          to === conversation.userA.virtualNumber.msisdn);
};

/**
 * Work out real number to virtual number mapping between users.
 */
VoiceProxy.prototype.getProxyRoute = function(from, to) {
  let proxyRoute = null;
  let conversation;
  for(let i = 0, l = this.conversations.length; i < l; ++i) {
    conversation = this.conversations[i];

    // Use to and from to determine the conversation
    const fromUserA = fromUserAToUserB(from, to, conversation);
    const fromUserB = fromUserBToUserA(from, to, conversation);

    if(fromUserA || fromUserB) {
      proxyRoute = {
        conversation: conversation,
```

```
        to: fromUserA? conversation.userB : conversation.userA,
        from: fromUserA? conversation.userA : conversation.userB
      };
      break;
    }
  }

  return proxyRoute;
};
```

Proxy the call

Proxy the call to the phone number the virtual number is associated with. The from number is always the virtual number, the to is a real phone number.



- Create an NCCO (Nexmo Call Control Object).
- This **NCCO** uses a **talk action** to read out some text.
- When the talk has completed, a connect action forwards the call to a real number.

Making Calls

Make an outbound call

This code snippet makes an outbound call and plays a text-to-speech message when the call is answered.

```
client = vonage.Client(
    application_id=VONAGE_APPLICATION_ID,
    private_key=VONAGE_APPLICATION_PRIVATE_KEY_PATH,
)

response = client.voice.create_call({
  'to': [{'type': 'phone', 'number': TO_NUMBER}],
  'from': {'type': 'phone', 'number': FROM_NUMBER},
  'answer_url': ['https://raw.githubusercontent.com/nexmo-community/ncco-
examples/gh-pages/text-to-speech.json']
})

print(response)
```

Make an outbound call with an NCCO

This code snippet makes an outbound call and plays a text-to-speech message when the call is answered. You don't need to run a server hosting an `answer_url` to run this code snippet, as you provide your NCCO as part of the request.

```
client = vonage.Client(
    application_id=VONAGE_APPLICATION_ID,
    private_key=VONAGE_APPLICATION_PRIVATE_KEY_PATH,
)

response = client.voice.create_call({
    'to': [{'type': 'phone', 'number': TO_NUMBER}],
    'from': {'type': 'phone', 'number': VONAGE_NUMBER},
    'ncco': [{
        'action': 'talk',
        'text': 'This is a text to speech call from Nexmo'
    }]
})

pprint(response)
```

Connect an inbound call

In this code snippet you see how to connect an inbound call to another person by making an outbound call.

```
from flask import Flask, jsonify

app = Flask(__name__)


@app.route("/webhooks/answer")
def answer_call():
    ncco = [
        {
            "action": "connect",
            "from": "VONAGE_NUMBER",
            "endpoint": [{
                "type": 'phone',
                "number": "YOUR_SECOND_NUMBER"
            }]
        }
    ]
    return jsonify(ncco)
```

Receiving Calls

Receive an inbound call

In this code snippet you see how to receive an inbound call.

```
from flask import Flask, request, jsonify

app = Flask(__name__)


@app.route("/webhooks/answer")
def answer_call():
    for param_key, param_value in request.args.items():
        print("{}: {}".format(param_key, param_value))

    from_ = request.args['from']

    return jsonify([
        {
            "action": "talk",
            "text": "Thank you for calling from " + " ".join(from_)
        }
    ])
```

Recording Calls

Record a call

A code snippets that shows how to answer an incoming call and set it up to record, then connect the call. When the call is completed, the eventUrl you specify in the record action of the NCCO will receive a webhook including the URL of the recording for download.

```
from pprint import pprint
from flask import Flask, request, jsonify

app = Flask(__name__)


@app.route("/webhooks/answer")
def answer_call():
    ncco = [
        {
            "action": "talk",
            "text": "Hi, we will shortly forward your call. This call is
recorded for quality assurance purposes."
        },
        {
            "action": "record",
            "eventUrl": ["https://demo.ngrok.io/webhooks/recordings"]
        },
        {
            "action": "connect",
            "eventUrl": ["https://demo.ngrok.io/webhooks/event"],
            "from": "VONAGE_NUMBER",
            "endpoint": [
                {
                    "type": "phone",
                    "number": "RECIPIENT_NUMBER"
                }
            ]
        }
    ]
    return jsonify(ncco)


@app.route("/webhooks/recordings", methods=['POST'])
def recordings():
    data = request.get_json()
    pprint(data)
    return "webhook received"
```

Record a message

A code snippet that shows how to record a conversation. Answer an incoming call and return an NCCO that includes a record action. When the call is complete, a webhook is sent to the eventUrl you specify. The webhook includes the URL of the recording.

```
from pprint import pprint
import http
from flask import Flask, request, jsonify

app = Flask(__name__)


@app.route("/webhooks/answer")
def answer_call():
    for param_key, param_value in request.args.items():
        print("{}: {}".format(param_key, param_value))
    recording_webhook_url = request.url_root + "webhooks/recording"
    ncco = [
        {
            "action": "talk",
            "text": "Please leave a message after the tone, then press the hash
key."
        },
        {
            "action": "record",
            "endOnKey": "#",
            "beepStart": "true",
            "eventUrl": [recording_webhook_url]
        },
        {
            "action": "talk",
            "text": "Thank you for your message."
        }
    ]
    return jsonify(ncco)


@app.route("/webhooks/recording", methods=['POST'])
def recording_webhook():
    pprint(request.get_json())
    return ('', http.HTTPStatus.NO_CONTENT)


if __name__ == '__main__':
    app.run(port=3000)
```

Retrieving Calls

Retrieve information for a call

A code snippet that shows how to retrieve information for a call. The call to retrieve information for is identified via a UUID.

```
client = vonage.Client(
        application_id=VONAGE_APPLICATION_ID,
        private_key=VONAGE_APPLICATION_PRIVATE_KEY_PATH,
    )

# Note call can be made to current call or a completed call
response = client.voice.get_call(VONAGE_CALL_UUID)
pprint(response)
```

Retrieve information for all calls

A code snippet that shows how to retrieve information for all calls.

```
client = vonage.Client(
    application_id=VONAGE_APPLICATION_ID,
    private_key=VONAGE_APPLICATION_PRIVATE_KEY_PATH,
)

NOW = datetime.utcnow()
DATE_END = NOW.replace(microsecond=0).isoformat()+"Z"
DATE_START = (NOW - timedelta(hours=24,
minutes=00)).replace(microsecond=0).isoformat()+"Z"

response = client.voice.get_calls(date_start=DATE_START, date_end=DATE_END)
calls = response['_embedded']['calls']
for call in calls:
    pprint(call)
```

Get a Conversation

In this code snippet you learn how to get a specified Conversation.

```
curl "https://api.nexmo.com/v0.3/conversations/$CONVERSATION_ID" \
     -H 'Authorization: Bearer '$JWT\
     -H 'Content-Type: application/json'
```

List Conversations

In this code snippet you learn how to list all Conversations.

```
# Gets a list of conversations from an application ID (app ID is in the JWT).
curl "https://api.nexmo.com/v0.3/conversations" \
     -H 'Authorization: Bearer '$JWT\
     -H 'Content-Type: application/json'
```

Connect to a WebSocket

Write your answer webhook

When Vonage receives an inbound call on your virtual number, it will make a request to your /webhooks/answer route. This route should accept an HTTP GET request and return a Nexmo Call Control Object (NCCO) that tells Vonage how to handle the call.

Your NCCO should use the text action to greet the caller, and the connect action to connect the call to your webhook endpoint:

```
from flask import Flask, request, jsonify
from flask_sock import Sock

app = Flask(__name__)
sock = Sock(app)


@app.route("/webhooks/answer")
def answer_call():
    ncco = [
        {
            "action": "talk",
            "text": "We will now connect you to the echo server, wait a moment
then start speaking.",
        },
        {
            "action": "connect",
            "from": "Vonage",
            "endpoint": [
                {
                    "type": "websocket",
                    "uri": f"wss://{request.host}/socket",
                    "content-type": "audio/l16;rate=16000",
                }
            ],
        },
    ]

    return jsonify(ncco)
```

The type of endpoint is websocket, the uri is the /socket route where your WebSocket server will be accessible and the content-type specifies the audio quality.

Write your event webhook

Implement a webhook that captures call events so that you can observe the lifecycle of the call in the console.

We won't use the request data in this tutorial, so your webhook should immediately return an HTTP 200 response (success):

```
@app.route("/webhooks/events", methods=["POST"])
def events():
    return "200"
```

Vonage makes a POST request to this endpoint every time the call status changes.

Create the WebSocket

Create a route handler for the /socket route. This listens for a message event which is raised every time the WebSocket receives audio from the call. Your application should respond by echoing the audio back to the caller with the send() method:

```
@sock.route("/socket", methods=["GET"])
def echo_socket(ws):
    while True:
        data = ws.receive()
        ws.send(data)
```

The WebSocket you created was extremely straightforward, but it was able to listen to the call audio and respond to it.

Play Audio into a WebSocket

You can use the Vonage Voice API to connect a call to a WebSocket, giving you a two-way stream of the call audio delivered over the WebSocket protocol in real-time.

you will connect an inbound call to a WebSocket endpoint.

The WebSocket server will stream an audio file into the call.

Create the WebSocket

First, handle the connection event so that you can report when your webhook server is online and ready to receive the call audio:

```
expressWs.getWss().on('connection', function (ws) {
  console.log('Websocket connection is open');
});
```

When writing audio to a Voice API WebSocket, the audio is expected in a specific format.

You will need a function that separates the binary audio data into arrays of the correct size:

```
function chunkArray(array, chunkSize) {
    var chunkedArray = [];
    for (var i = 0; i < array.length; i += chunkSize)
        chunkedArray.push(array.slice(i, i + chunkSize));
    return chunkedArray;
}
```

Create a route handler for the /socket route. When the WebSocket is connected this route will get called:

```
app.ws('/socket', (ws, req) => {
    const wav = new WaveFile(fs.readFileSync("./sound.wav"));
    wav.toSampleRate(16000);
    wav.toBitDepth("16");

    const samples = chunkArray(wav.getSamples()[0], 320);
    for (var index = 0; index < samples.length; ++index) {
        ws.send(Uint16Array.from(samples[index]).buffer);
    }
})
```

Vonage will only buffer 1024 messages which should be enough for around 20 seconds of audio, if your file is longer than this you should implement a delay of 18-19ms between each message.

Managing Users

Create a User

In this code snippet you learn how to create a User.

```
vonage.users.create({
"name": USER_NAME,
"display_name": USER_DISPLAY_NAME}, (error, result) => {
    if(error) {
        console.error(error);
    }
    else {
        console.log(result);
    }
});
```

Delete a User

In this code snippet you learn how to delete a User.

```
vonage.users.delete(USER_ID, (error, result) => {
    if(error) {
        console.error(error);
    }
    else {
        console.log(result);
    }
});
```

Get a User

In this code snippet you learn how to get a User.

```
vonage.users.get(USER_ID, (error, result) => {
  if(error) {
      console.error(error);
  }
  else {
      console.log(result);
  }
});
```

Update a User

In this code snippet you learn how to update a User's details.

```
vonage.users.update(USER_ID, {
"name": USER_NEW_NAME,
"display_name": USER_NEW_DISPLAY_NAME}, (error, result) => {
  if(error) {
      console.error(error);
  }
  else {
      console.log(result);
  }
});
```

List Users

In this code snippet you learn how to get a list Users associated with an Application.

```
vonage.users.get({}, (error, result) => {
  if(error) {
      console.error(error);
  }
  else {
      console.log(result);
  }
});
```

Voice API

Calls

Fetch, Create and Modify voice calls

Available Operations:

GET - Get details of your calls

https://api.nexmo.com/v1/calls/

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Query Parameters

Fields	Types
status	string
date_start	string (date-time)
date_end	string (date-time)
page_size	integer
record_index	integer
order	string
conversation_uuid	string (uuid)

Response


```
{
   "count": 100,
   "page_size": 10,
   "record_index": 0,
   "_links": {
      "self": {
         "href": "/calls?page_size=10&record_index=20&order=asc"
      }
   },
   "_embedded": {
      "calls": [
         {
            "_links": {
               "self": {
                  "href": "/calls/63f61863-4a51-4f6b-86e1-46edebcf9356"
               }
            },
            "uuid": "63f61863-4a51-4f6b-86e1-46edebcf9356",
            "conversation_uuid": "CON-f972836a-550f-45fa-956c-12a2ab5b7d22",
            "to": {
```

```
            "type": "phone",
            "number": "447700900000"
         },
         "from": {
            "type": "phone",
            "number": "447700900001"
         },
         "status": "completed",
         "direction": "outbound",
         "rate": "0.39",
         "price": "23.40",
         "duration": "60",
         "start_time": "2020-01-01 12:00:00",
         "end_time": "2020-01-01 12:00:00",
         "network": "65512"
      }
   ]
}
```


POST - Create an outbound call

https://api.nexmo.com/v1/calls/

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Request Body – With NCCO – Required Fields only

Fields	Type
ncco	array
to	array
type	string
number	string

Response


```
{
   "uuid": "63f61863-4a51-4f6b-86e1-46edebcf9356",
   "status": "completed",
   "direction": "outbound",
   "conversation_uuid": "CON-f972836a-550f-45fa-956c-12a2ab5b7d22"
}
```

GET - Get detail of a specific call

https://api.nexmo.com/v1/calls/:uuid

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Path Parameters

Field	Type
uuid	string

Response


```
{
   "_links": {
      "self": {
         "href": "/calls/63f61863-4a51-4f6b-86e1-46edebcf9356"
      }
   },
   "uuid": "63f61863-4a51-4f6b-86e1-46edebcf9356",
   "conversation_uuid": "CON-f972836a-550f-45fa-956c-12a2ab5b7d22",
   "to": {
      "type": "phone",
      "number": "447700900000"
   },
   "from": {
      "type": "phone",
      "number": "447700900001"
   },
   "status": "completed",
   "direction": "outbound",
   "rate": "0.39",
   "price": "23.40",
   "duration": "60",
   "start_time": "2020-01-01 12:00:00",
   "end_time": "2020-01-01 12:00:00",
   "network": "65512"
}
```

PUT - Modify an in progress call

https://api.nexmo.com/v1/calls/:uuid

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Path Parameters

Field	Type
uuid	string

Request Body

Fields	Type
action	string
destination	object
type	string
ncco	array

Response

Status Code	Content
204	No content
401	Unauthorized
404	Not found

Stream Audio

Start or stop streaming audio in to an active call

Available Operations

PUT - Play an audio file into a call

https://api.nexmo.com/v1/calls/:uuid/stream

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Path Parameters

Field	Type
uuid	string

Request Body

Fields	Types
stream_url	array
loop	integer
level	string

Response

```
{
   "message": "Stream started",
   "uuid": "63f61863-4a51-4f6b-86e1-46edebcf9356"
}
```

DELETE - Stop playing an audio file into a call

https://api.nexmo.com/v1/calls/:uuid/stream

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Path Parameters

Field	Type
uuid	string

Response

```
{
   "message": "Stream stopped",
   "uuid": "63f61863-4a51-4f6b-86e1-46edebcf9356"
}
```

Play TTS

Start or stop playing Text to Speech in to an active call

Available Operations

PUT - Play text to speech into a call

https://api.nexmo.com/v1/calls/:uuid/talk

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Path Parameters

Field	Type
uuid	string

Request Body

Fields	Types
text	string
language	string
style	integer
premium	boolean
voice_name	string
loop	integer
level	string

Response

```
{
   "message": "Talk started",
   "uuid": "63f61863-4a51-4f6b-86e1-46edebcf9356"
}
```

DELETE - Stop text to speech in a call

https://api.nexmo.com/v1/calls/:uuid/talk

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Path Parameters

Field	Type
uuid	string

Response

```
{
   "message": "Talk stopped",
   "uuid": "63f61863-4a51-4f6b-86e1-46edebcf9356"
}
```

Play DTMF

Play DTMF tones in to an active call

Available Operations

PUT - Play DTMF tones into a call

https://api.nexmo.com/v1/calls/:uuid/dtmf

Authentication

Key	Description	Where	Example
Authorization	Your JSON web token	Headers	Bearer <JWT>

Path Parameters

Field	Type
uuid	string

Request Body

Field	Type
digits	string

Response

```
{
   "message": "DTMF sent",
   "uuid": "63f61863-4a51-4f6b-86e1-46edebcf9356"
}
```