Table of Contents

Learn how to enable natural and contextual interaction within your applications with Cognitive Services. Quick start tutorials and API references help you incorporate artificial intelligence capabilities for text, speech, vision, and search.

- **Learn about Cognitive Services**
- **Get started with the Computer Vision API**
- **Get started with the Face API**
- **Get started with the Bing Web Search API**
- **Get started with the Custom Speech Service API**
- **Get started with Language Understanding Intelligent Services (LUIS)**
- **Cognitive Services video library**

Bing Speech API overview

7/25/2017 • 2 min to read • Edit Online

Microsoft's *Speech to Text* and *Text to Speech* cloud offerings help you put speech to work in your application. Microsoft's Speech APIs can transcribe speech *to* text and can generate speech *from* text. These APIs enable you to create powerful experiences that delight your users.

- **Speech to Text** APIs convert human speech to text that can be used as input or commands to control your application.
- **Text to Speech** APIs convert text to audio streams that can be played back to the user of your application.

Speech to text (speech recognition)

The *Speech to Text* APIs *transcribe* audio streams into text that your application can display to the user or act upon as command input. The *Speech To Text* APIs come in two flavors.

- A REST API, useful for apps that need to convert short spoken commands to text but do not need simultaneous user feedback. The REST API uses HTTP chunked-transfer encoding to send the audio bytes to the service.
- A WebSocket API, useful for apps need an improved user experience by using the power of the full-duplex WebSocket connection. Apps using this API get access to advanced features like speech recognition hypotheses. This API choice is also better for apps that need to transcribe longer audio passages.

Both *Speech to Text* APIs enrich the transcribed text by adding capitalization and punctuation, masking profanity, and text normalization.

Comparing API options for speech recognition

FEATURE	WEBSOCKET API	REST API
Speech hypotheses	Yes	No
Continuous recognition	Yes	No
Maximum audio input	10 minutes of audio	15 seconds of audio
Service detects when speech ends	Yes	No
Subscription key authorization	Yes	No

Speech recognition modes

Microsoft's *Speech to Text* APIs support multiple modes of speech recognition. Choose the mode that produces the best recognition results for your application.

MODE	DESCRIPTION
interactive	"Command and control" recognition for interactive user application scenarios. Users speak short phrases intended as commands to an application.

MODE	DESCRIPTION
dictation	Continuous recognition for dictation scenarios. Users speak longer sentences that are displayed as text. Users adopt a more formal speaking style.
conversation	Continuous recognition for transcribing conversations between humans. Users adopt a less formal speaking style and may alternate between longer sentences and shorter phrases.

For more information, see Recognition Modes in the API reference.

Speech recognition supported languages

The *Speech to Text* APIs support many spoken languages in multiple dialects. For the full list of supported languages in each recognition mode, see Recognition Languages.

Text to speech (speech synthesis)

Text to Speech APIs use REST to convert structured text to an audio stream. The APIs provide fast text to speech conversion in various voices and languages. For the full list of supported languages and voices, see Supported Locales and Voice Fonts.

Text to speech API

The maximum amount of audio returned for any single request is 15 seconds.

Get Started with Speech Recognition using REST API

6/28/2017 • 5 min to read • Edit Online

With Bing Speech API you can develop applications using REST API to convert spoken audio to text. This article will help you to do speech recognition via REST end point.

Prerequisites

Subscribe to Speech API and get a free trial subscription key

To access the REST end point, you must subscribe to Speech API which is part of Microsoft Cognitive Services (previously Project Oxford). After subscribing, you will have the necessary subscription keys to execute this operation. Both the primary and secondary keys can be used. For subscription and key management details, see Subscriptions.

Precorded audio file

Record a short audio file of you saying something short (e.g.: "*What is the weather like today?*" or "*Find funny movies to watch.*") You will pass this audio to the Bing Speech API via the REST end point to have it transcribe into text. Or, you can use the microphone at the time of the request.

> **NOTE**
>
> The Speech Recognition API supports audio/wav using the following codecs:
>
> - PCM single channel

Getting started

To use Speech API REST end point, the process is as follows:

1. Authenticate and get a JSON Web Token (JWT) from the token service.
2. Set the proper request header and send the request to Bing Speech API REST end point.
3. Parse the response to get your transcribed text.

The sections following will provide more details.

Authentication

To access the REST endpoint, you need a valid OAuth token. To get this token, you must have a subscription key from the Speech API. When you request for a token, the token service will send the access token back as a JSON Web Token (JWT). The JWT access token is passed through in the Speech request header. The token has an expiry of 10 minutes. The recommendation is to look into the JWT token and check the expiry time instead of hard-coding it to 10 minutes using the Expiration JwtSecurityToken property.

The token service URI is located here:

```
https://api.cognitive.microsoft.com/sts/v1.0/issueToken
```

The code below is an example implementation in C# for how to handle authentication:

```
/*
 * This class demonstrates how to get a valid O-auth token
```

```
 * This class demonstrates how to get a valid O-auth token.
 */
public class Authentication
{
    public static readonly string FetchTokenUri = "https://api.cognitive.microsoft.com/sts/v1.0";
    private string subscriptionKey;
    private string token;
    private Timer accessTokenRenewer;

    //Access token expires every 10 minutes. Renew it every 9 minutes.
    private const int RefreshTokenDuration = 9;

    public Authentication(string subscriptionKey)
    {
        this.subscriptionKey = subscriptionKey;
        this.token = FetchToken(FetchTokenUri, subscriptionKey).Result;

        // renew the token on set duration.
        accessTokenRenewer = new Timer(new TimerCallback(OnTokenExpiredCallback),
                                       this,
                                       TimeSpan.FromMinutes(RefreshTokenDuration),
                                       TimeSpan.FromMilliseconds(-1));
    }

    public string GetAccessToken()
    {
        return this.token;
    }

    private void RenewAccessToken()
    {
        this.token = FetchToken(FetchTokenUri, this.subscriptionKey).Result;
        Console.WriteLine("Renewed token.");
    }

    private void OnTokenExpiredCallback(object stateInfo)
    {
        try
        {
            RenewAccessToken();
        }
        catch (Exception ex)
        {
            Console.WriteLine(string.Format("Failed renewing access token. Details: {0}", ex.Message));
        }
        finally
        {
            try
            {
                accessTokenRenewer.Change(TimeSpan.FromMinutes(RefreshTokenDuration), TimeSpan.FromMilliseconds(-1));
            }
            catch (Exception ex)
            {
                Console.WriteLine(string.Format("Failed to reschedule the timer to renew access token. Details: {0}", ex.Message));
            }
        }
    }

    private async Task<string> FetchToken(string fetchUri, string subscriptionKey)
    {
        using (var client = new HttpClient())
        {
            client.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", subscriptionKey);
            UriBuilder uriBuilder = new UriBuilder(fetchUri);
            uriBuilder.Path += "/issueToken";

            var result = await client.PostAsync(uriBuilder.Uri.AbsoluteUri, null);
            Console.WriteLine("Token Uri: {0}", uriBuilder.Uri.AbsoluteUri);
            return await result.Content.ReadAsStringAsync();
        }
```

```
        }
    }
}
```

The `FetchToken` method sends the request and the authentication payload is as follows:

```
POST https://api.cognitive.microsoft.com/sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: <GUID>
Host: api.cognitive.microsoft.com
Content-Length: 0
Connection: Keep-Alive
```

REST end point

The URL hostname for all requests is `speech.platform.bing.com`. For example,

```
https://speech.platform.bing.com/speech/recognition/interactive/cognitiveservices/v1?language=it-IT
```

Recognition Mode

You specify the *recognition mode* as part of the URL path for the Microsoft Speech Service. The following recognition modes are supported.

MODE	PATH	URL EXAMPLE
Interactive/Command	/speech/recognition/interactive/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/interactive/cognitiveservices/v1?language=pt-BR
Conversation	/speech/recognition/conversation/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
Dictation	/speech/recognition/dictation/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/dictation/cognitiveservices/v1?language=fr-FR

Request headers

To send the request to the REST end point, create a **HttpWebRequest** object and set the request headers. For example, set `Method="POST"`, set `Host=@"speech.platform.bing.com"`, and set the access token as follows `Headers["Authorization"] = "Bearer " + token;`. The code snippet below shows a sample of a request header.

```
HttpWebRequest request = null;
request = (HttpWebRequest)HttpWebRequest.Create(requestUri);
request.SendChunked = true;
request.Accept = @"application/json;text/xml";
request.Method = "POST";
request.ProtocolVersion = HttpVersion.Version11;
request.Host = @"speech.platform.bing.com";
request.ContentType = @"audio/wav; codec=""audio/pcm""; samplerate=16000";
request.Headers["Authorization"] = "Bearer " + token;
```

The follow is a sample request payload:

```
POST https://speech.platform.bing.com/speech/recognition/interactive/cognitiveservices/v1?language=en-US&format=detailed&requestid=39530efe-5677-416a-98b0-93e13ec93c2b HTTP/1.1
Accept: application/json;text/xml
Content-Type: audio/wav; codec="audio/pcm"; samplerate=16000
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzY29wZSI6Imh0dHBzOi8vc3BlZWNoLnBsYXRmb3JtLmJpbmcuY29tIiwic3Vic2NyaXB0aW9uLWlkIjoiMWRjYWQxZTQzZWZlNDM2MmIzMjg2ZWY2OTIzYTA5MjYiLCJwcm9kdWN0LWlkIjoiQmluZy5TcGVlY2guRjAiLCJjb2duaXRpdmUtc2VydmljZXMtZW5kcG9pbnQiOiJodHRwczovL2FwaS5jb2duaXRpdmUubWljcm9zb2Z0LmNvbS9pbnRlcm5hbC92MS4wLyIsImF6dXJlLXJlc291cmNlLWlkIjoiL3N1YnNjcmlwdGlvbnMvYTM0Y2FkYmYtNTU5My00ZWYxLWI0MjItMDJhMDMyNmQ2NmZkL3Jlc291cmNlR3JvdXBzL1Rlc3QvcHJvdmlkZXJzL01pY3Jvc29mdC5Db2duaXRpdmVTZXJ2aWNlcy9hY2NvdW50cy9UZXN0U1BlZWNoIiwiaXNzIjoidXJuOm1zLnNwZWVjaCIsImV4cCI6MTQ5MzQyOTE2OX0._Bhx7nneMto2gjAAwmIO6eiSejQ2Nqhd8xFl0odjk40
Host: speech.platform.bing.com
Transfer-Encoding: chunked
Expect: 100-continue
```

Chunked transfer encoding

Bing Speech API supports chuncked transfer encoding for efficient audio streaming. To transcribe speech to text, you can send the audio as one whole chunk or you can chop the audio into small chunks. For efficient audio transcription it is recommended that you use chunked transfer encoding to stream the audio to the service. Other implementations may result in higher user-perceived latency.

> **NOTE**
>
> You may not upload more than 10 seconds of audio in any one request and the total request duration cannot exceed 14 seconds.

The code snippet below shows an example of an audio file being chunked into 1024 byte chunks.

```
using (fs = new FileStream(audioFile, FileMode.Open, FileAccess.Read))
{

  /*
  * Open a request stream and write 1024 byte chunks in the stream one at a time.
  */
  byte[] buffer = null;
  int bytesRead = 0;
  using (Stream requestStream = request.GetRequestStream())
  {
    /*
    * Read 1024 raw bytes from the input audio file.
    */
    buffer = new Byte[checked((uint)Math.Min(1024, (int)fs.Length))];
    while ((bytesRead = fs.Read(buffer, 0, buffer.Length)) != 0)
    {
      requestStream.Write(buffer, 0, bytesRead);
    }

    // Flush
    requestStream.Flush();
  }
}
```

Speech recognition response

After processing the request, Bing Speech API returns the results in a response as JSON format. The code snippet below shows an example of how you can read the response from the stream:

```
/*
* Get the response from the service.
*/
Console.WriteLine("Response:");
using (WebResponse response = request.GetResponse())
{
    Console.WriteLine(((HttpWebResponse)response).StatusCode);

    using (StreamReader sr = new StreamReader(response.GetResponseStream()))
    {
        responseString = sr.ReadToEnd();
    }

    Console.WriteLine(responseString);
    Console.ReadLine();
}
```

The following is a sample JSON response:

```
OK
{
    "RecognitionStatus": "Success",
    "Offset": 22500000,
    "Duration": 21000000,
    "NBest": [{
        "Confidence": 0.941552162,
        "Lexical": "find a funny movie to watch",
        "ITN": "find a funny movie to watch",
        "MaskedITN": "find a funny movie to watch",
        "Display": "Find a funny movie to watch."
    }]
}
```

Get started with the Speech REST API in cURL

8/7/2017 • 1 min to read • Edit Online

You can use the Bing Speech Recognition API with cURL to convert spoken audio to text by sending audio to Microsoft servers in the cloud. The examples in this article contain bash commands that demonstrate the use of Microsoft Cognitive Services (formerly Project Oxford) Speech To Text API using cURL.

The Speech Recognition web example demonstrates the following features when you use a .wav file or an external microphone input:

- Access-token generation
- Short-form recognition

This example assumes that cURL is available in your bash environment.

Prerequisites

- Platform requirements: The examples shown in the following sections were developed in bash. They also work in Git Bash, zsh, and other shells.
- Subscribe to Speech API and get a free trial subscription key. Speech API is part of Microsoft Cognitive Services. For subscription and key management details, see Subscriptions. You can use both the primary and secondary keys in this tutorial.

Step 1: Generate an access token

1. Replace **your_subscription_key** with your own subscription key, and then run the command in bash.

```
curl -v -X POST "https://api.cognitive.microsoft.com/sts/v1.0/issueToken" -H "Content-type: application/x-www-form-urlencoded" -H "Content-Length: 0" -H "Ocp-Apim-Subscription-Key: your_subscription_key"
```

2. The response is a string with the JSON Web Token (JWT) access token:

```
JWT access token
```

Step 2: Upload the audio binary

1. Replace **your_locale**, **your_format**, and **your_guid** in accordance with your own application.
2. Replace **your_access_token** with the JWT access token that you retrieved in the Step 1: Generate an access token section.
3. Replace **your_wave_file** with the actual wave file.
4. In bash, run the following command:

```
curl -v -X POST "https://speech.platform.bing.com/speech/recognition/interactive/cognitiveservices/v1?language=pt-BR&locale=your_locale&format=your_format&requestid=your_guid" -H "Transfer-Encoding: chunked" -H 'Authorization: Bearer your_access_token' -H 'Content-type: audio/wav; codec="audio/pcm"; samplerate=16000' --data-binary @your_wave_file
```

5. Parse the successful recognition response or error response.

Recognition mode

Specify the *recognition mode* as part of the URL path for Microsoft Speech Service. The following recognition modes are supported.

MODE	PATH	URL EXAMPLE
Interactive/Command	/speech/recognition/interactive/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/interactive/cognitiveservices/v1?language=pt-BR
Conversation	/speech/recognition/conversation/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
Dictation	/speech/recognition/dictation/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/dictation/cognitiveservices/v1?language=fr-FR

For questions, feedback, or suggestions about Microsoft Cognitive Services, feel free to reach out to us directly at Cognitive Services UserVoice Forum.

License

Get Started with Bing Speech API using web socket

8/21/2017 • 2 min to read • Edit Online

Bing Speech Recognition API allows you to develop applications using Web socket to convert spoken audio to text. One major advantage of using web socket, compared to HTTP connections, is that web socket connections last a long time. This allows you to keep on talking and receiving transcribed text simultaneously. This article will help you get started in using Bing Speech API with web socket.

Prerequisites

Subscribe to Speech API and get a free trial subscription key

To work with Bing Speech API, you must have a subscription key. If you don't have a subscription key already, get one here: Subscriptions.

Get started

To get started with Bing Speech Recognition API using web socket, we have created a working HTML/JS sample ready for you to try. The steps below will walk you through how to get the source code and how to run the sample. After getting a subscription key you can either pass it in directly into the speech endpoint or use authentication endpoint as shown below to get a token and pass that in to speech endpoint using the SDK.

```
POST https://api.cognitive.microsoft.com/sts/v1.0/issueToken HTTP/1.1
Ocp-Apim-Subscription-Key: <GUID>
Host: api.cognitive.microsoft.com
Content-Length: 0
Connection: Keep-Alive
```

To run the sample, do the follow:

1. Download a copy of the HTML file and the JavaScript file from the SpeechToText-WebSockets-Javascript repository: Sample.html and speech.browser.sdk.js
2. Open the **Sample.html** file in a text editor and edit the follow two lines: Replace 'YOUR_BING_SPEECH_API_KEY' with your subscription key. Replace '....\distrib\speech.browser.sdk.js' with corrected path to the SDK JavaScript file.

   > **NOTE**
   > To work with Bing Speech Recognition API, all you need is a subscription key. However, the Speech API for JavaScript using web socket also support authentication token. If you have an authentication token and preferred to use that then do the following. Replace 'let authentication = new SR.CognitiveSubscriptionKeyAuthentication(subscriptionKey);' statement with this 'let authentication = new SR.CognitiveTokenAuthentication(fetchCallback, fetchOnExpiryCallback);'.

3. Open **Sample.html** in a web browser.
4. Click the **Start** button. This will initialize the sample and turns on the microphone. Grant the browser access to your microphone if the browser asks for permission.
5. Start talking. Your transcribed text appears after the **Current hypothesis:** label. The text area will display the JSON payload of the transcribed audio.

Remarks

- The Bing Speech Recognition API for web socket supports three recognition modes. You can switch the mode by updating the **Setup()** function found in the **Sample.html** file. The sample set the mode to **Interactive** by default. To change the mode, update the parameter 'SR.RecognitionMode.Interactive' to another mode, for example 'SR.RecognitionMode.Conversation'.
- For a complete list of supported languages, see Supported Languages

See also

- Web socket sample repository: SpeechToText-WebSockets-Javascript
- Bing Speech API via REST: Get started with REST API

Getting Started with Bing Speech Recognition in C# for .Net Windows

6/27/2017 • 5 min to read • Edit Online

Develop a basic Windows application that uses Bing Speech Recognition API to convert spoken audio to text by sending audio to Microsoft's servers in the cloud. Using the Client Library allows for real-time streaming, which means that at the same time your client application sends audio to the service, it simultaneously and asynchronously receives partial recognition results back. This page describes use of the Client Library, which currently supports speech in seven languages, the example below defaults to American English, "en-US". For client library API reference, see Microsoft Bing Speech SDK.

Prerequisites

Platform Requirements

The below example has been developed for Windows 8+ and .NET Framework 4.5+ using Visual Studio 2015, Community Edition.

Get the Client Library and Example

You may download the Speech API client library and example through SDK. The downloaded zip file needs to be extracted to a folder of your choice, many users choose the Visual Studio 2015 folder.

Subscribe to Speech API and Get a Free Trial Subscription Key

Before creating the example, you must subscribe to Speech API which is part of Microsoft Cognitive Services (previously Project Oxford). For subscription and key management details, see Subscriptions. Both the primary and secondary key can be used in this tutorial.

Step 1: Install the Example Application

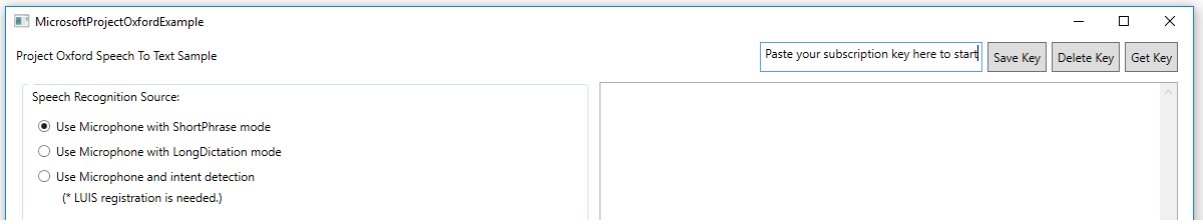
1. Start Microsoft Visual Studio 2015 and click **File**, select **Open**, then **Project/Solution**.
2. Browse to the folder where you saved the downloaded Speech API files. Click on **Speech**, then **Windows**, and then the **Sample-WPF** folder.
3. Double-click to open the Visual Studio 2015 Solution (.sln) file named **SpeechToText-WPF-Samples.sln**. This will open the solution in Visual Studio.

Step 2: Build the Example Application

1. Press Ctrl+Shift+B, or click **Build** on the ribbon menu, then select **Build Solution**.

Step 3: Run the Example Application

1. After the build is complete, press **F5** or click **Start** on the ribbon menu to run the example.
2. Locate the **Project Oxford Speech to Text** window with the **text edit box** reading **"Paste your subscription key here to start"**. Paste your subscription key into the text box as shown in below screenshot. You may choose to persist your subscription key on your PC or laptop by clicking the **Save Key** button. When you want to delete the subscription key from the system, click **Delete Key** to remove it from your PC or laptop.



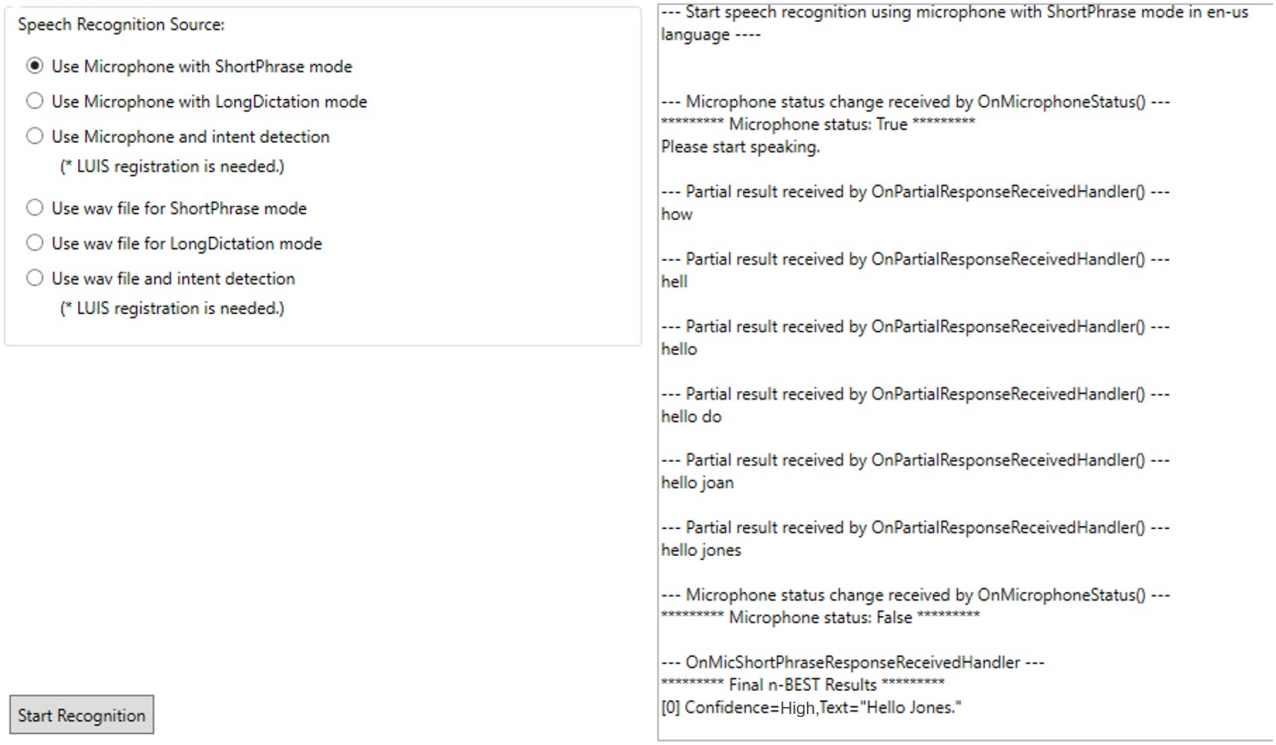
3. Under **Speech Recognition Source** choose one of the six speech sources, which fall into two main input categories.

   - Using your computer's microphone, or an attached microphone, to capture speech.
   - Playing an audio file.

Each category has three recognition modes.

- **ShortPhrase Mode**: An utterance up to 15 seconds long. As data is sent to the server, the client will receive multiple partial results and one final multiple N-best choice result.
- **LongDictation Mode**: An utterance up to 2 minutes long. As data is sent to the server, the client will receive multiple partial results and multiple final results, based on where the server indicates sentence pauses.
- **Intent Detection**: The server returns additional structured information about the speech input. To use Intent you will need to first train a model. See details here.

There are example audio files to be used with this example application. You find the files in the repository you downloaded with this example under **SpeechToText**, in the **Windows** folder, under **samples**, in the **SpeechRecognitionServiceExample** folder. These example audio files will run automatically if no other files are chosen when selecting the **Use wav file for Shortphrase mode** or **Use wav file for Longdictation mode** as your speech input. Currently only wav audio format is supported.



Review and Learn

Events

Partial Results Event:

This event gets called every time the Speech Recognition Server has an idea of what the speaker might be saying – even before he or she has finished speaking (if you are using the Microphone Client) or have finished transferring

data (if you are using the Data Client).

Intent Event:

Called on WithIntent clients (only in ShortPhrase mode) after the final reco result has been parsed into structured JSON intent.

Result Event:

When you have finished speaking (in ShortPhrase mode), this event is called. You will be provided with n-best choices for the result. In LongDictation mode, the handler associated with this event will be called multiple times, based on where the server thinks sentence pauses are.

Eventhandlers are already pointed out in the code in form of code comments.

RETURN FORMAT	DESCRIPTION
LexicalForm	This form is optimal for use by applications that need raw, unprocessed speech recognition results.
DisplayText	The recognized phrase with inverse text normalization, capitalization, punctuation and profanity masking applied. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to a user.
Inverse Text Normalization (ITN) has been applied	An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that display the speech recognition results to a user.
InverseTextNormalizationResult	Inverse text normalization (ITN) converts phrases like "one two three four" to a normalized form such as "1234". Another example is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that interpret the speech recognition results as commands or perform queries based on the recognized text.
MaskedInverseTextNormalizationResult	The recognized phrase with inverse text normalization and profanity masking applied, but no capitalization or punctuation. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to a user. Inverse Text Normalization (ITN) has also been applied. An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that use the unmasked ITN results but also need to display the command or query to the user.

Related Topics

- Get started with Bing Speech Recognition and/or intent in Java on Android
- Get started with Bing Speech Recognition and/or intent in Objective C on iOS
- Get started with Bing Speech API in JavaScript
- Get started with Bing Speech API in cURL

Get Started with Bing Speech Recognition Service Library in C# for .Net Windows

6/27/2017 • 5 min to read • Edit Online

With Microsoft Speech Recognition Service Library, your service can utilize the power of Microsoft Speech transcription cloud to convert spoken language to text. This service-to-service library works in real-time so your client app can send audio to servers in the cloud and start receiving partial recognition results back simultaneously and asynchronously. For library API reference, see the Microsoft Bing Speech SDK.

Speech Recognition Service C# Sample

This section describes how to install, build, and run the C# sample application.

Prerequisites

- **Platform Requirements**: The below example has been developed for Windows 8+ and .NET 4.5+ Framework using Visual Studio 2015, Community Edition.
- **Get the Service Library and Sample Application**: The library is available through a Nuget Package. You may clone the sample through Github.
- **Subscribe to Speech API and Get a Free Trial Subscription Key**: Before creating the example, you must subscribe to Speech API which is part of Microsoft Cognitive Services (previously Project Oxford). For subscription and key management details, see Subscriptions. Both the primary and secondary key can be used in this tutorial.

1. Start Microsoft Visual Studio 2015 and click **File**, select **Open**, then **Project/Solution**.
2. Double-click to open the Visual Studio 2015 Solution (.sln) file named **SpeechClient.sln**. This will open the solution in Visual Studio.

Press Ctrl+Shift+B, or click **Build** on the ribbon menu, then select **Build Solution**.

1. After the build is complete, press **F5** or click **Start** on the ribbon menu to run the example.
2. Open the output directory for the sample (**SpeechClientSample\bin\Debug**), press **Shift+Right Click**, press **Open command window here**.
3. Run **SpeechClientSample.exe** with the following arguments:
   - Arg[0]: Specify an input audio wav file.
   - Arg[1]: Specify the audio locale.
   - Arg[2]: Specify the service uri.
   - Arg[3]: Specify the subscription key to access the Speech Recognition Service.

- **ShortPhrase mode:** an utterance up to 15 seconds long. As data is sent to the server, the client will receive multiple partial results and one final best result.
- **LongDictation mode:** an utterance up to 10 minutes long. As data is sent to the server, the client will receive multiple partial results and multiple final results, based on where the server indicates sentence pauses.

RECOGNITION MODE	SERVICE URI
Short-Form	wss://speech.platform.bing.com/api/service/recognition

RECOGNITION MODE	SERVICE URI
Long-Form	wss://speech.platform.bing.com/api/service/recognition/continuous

The Voice API supports audio/wav using the following codecs:

- PCM single channel
- Siren
- SirenSR

To create a SpeechClient, you need to first create a Preferences object. The Preferences object is a set of parameters that configures the behavior of the speech service. It consists of the following fields:

- **SpeechLanguage:** The locale of the audio being sent to the speech service.
- **ServiceUri:** The endpoint use to call the speech service.
- **AuthorizationProvider:** An IAuthorizationProvider implemetation used to fetch tokens in order to access the speech service. Although the sample provides a Cognitive Services authorization provider, it is highly recommended to create your own implementation to handle token caching.
- **EnableAudioBuffering:** An advanced option, please see Connection Management

The SpeechInput object consists of 2 fields:

- **Audio**: A stream implementation of your choice that the SDK will pull audio from. Please note that this could be any Stream that supports reading. **Note**: the SDK detects the end of of the stream when it the stream returns **0** when attempting to read from it.
- **RequestMetadata**: Metadata about the speech request. For more details refer to the documentation.

Once you have instantiated a SpeechClient and SpeechInput objects, use RecognizeAsync to make a request to the speech service.
var task = speechClient.RecognizeAsync(speechInput);
The task returned by RecognizeAsync completes once the request completes. The last RecognitionResult that the server thinks is the end of the recognition. The task can Fault if the server or the SDK fails unexpectedly.

Partial Results Event:
This event gets called every time the Speech Recognition Server has an idea of what the speaker might be saying – even before the user has finished speaking (if you are using the Microphone Client) or have finish transferring data (if you are using the Data Client). You can subscribe to the event using
SpeechClient.SubscribeToPartialResult();
Or use the generic events subscription method
SpeechClient.SubscribeTo();

RETURN FORMAT	DESCRIPTION
LexicalForm	This form is optimal for use by applications that need raw, unprocessed speech recognition results.
DisplayText	The recognized phrase with inverse text normalization, capitalization, punctuation and profanity masking applied. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to a user.

RETURN FORMAT	DESCRIPTION
Confidence	Indicates the level of confidence the recognized phrase represents the audio associated as defined by the Speech Recognition Server.
MediaTime	The current time relative to the start of the audio stream (In 100-nanosecond units of time).
MediaDuration	The current phrase duration/length relative to the audio segment (In 100-nanosecond units of time).

Result Event:

When you have finished speaking (in ShortPhrase mode), this event is called. You will be provided with n-best choices for the result. In LongDictation mode, the event can be called multiple times, based on where the server thinks sentence pauses are. You can subscribe to the event using
SpeechClient.SubscribeToRecognitionResult();
Or Use the generic events subscription method
SpeechClient.SubscribeTo();

RETURN FORMAT	DESCRIPTION
RecognitionStatus	The status on how the recognition was produced. For example, was it produced as a result of successful recognition, or as a result of canceling the connection, etc.
Phrases	The set of n-best recognized phrases with the recognition confidence. Refer to the above table for phrase format.

Speech Response example:

```
--- Partial result received by OnPartialResult
---what
--- Partial result received by OnPartialResult
--what's
--- Partial result received by OnPartialResult
---what's the web
--- Partial result received by OnPartialResult
---what's the weather like
---***** Phrase Recognition Status = [Success]
***What's the weather like? (Confidence:High)
What's the weather like? (Confidence:High)
```

The APIs utilizes a single web-socket connection per request. For optimal user experience, the SDK will attempt to reconnect to the speech service and start the recognition from the last RecognitionResult that it received. For example, if the audio request is 2 minutes long and the SDK received a RecognitionEvent at the 1 minute mark, then a network failure occurred after 5 seconds, the SDK will start a new connection starting from the 1 minute mark.

> **NOTE**
> The SDK does not seek back to the 1 minute mark, as the Stream may not support seeking. Instead the SDK keep internal buffer that it uses to buffer the audio and clears the buffer as it received RecognitionResult events.

By default, the SDK buffers audio so it can recover when a network interrupt occurs. In some scenario where it is

preferable to discard the audio lost during the network disconnect and restart the connection where the stream at due to performance considerations, it is best to disable audio buffering by setting **EnableAudioBuffering** in the Preferences object to **false**.

Get Started with Bing Speech Recognition API in Objective C on iOS

6/27/2017 • 7 min to read • Edit Online

With Bing Speech Recognition API you can develop iOS applications that leverage Microsoft cloud servers to convert spoken audio to text. The API supports real-time streaming, so your application can simultaneously and asynchronously receive partial recognition results at the same time it is sending audio to the service

This article uses a sample application to demonstrate the basics of getting started with the Bing Speech Recognition API to develop an iOS application. For a complete API reference, see Speech SDK Client Library Reference.

Prerequisites

Platform requirements

Make sure Mac XCode IDE is installed.

Get the client library and example

You may download the Speech API client library and example for iOS through https SDK. The downloaded zip file needs to be extracted to a folder of your choice. Install the .pkg file on your Mac. The .pkg file will install onto your Mac hard drive in the root (or personal) Documents directory under **SpeechSDK**. Inside the folder there is both a fully buildable example and an SDK library. The buildable example can be found in the **samples\SpeechRecognitionServerExample** directory and the library can be found at the **SpeechSDK\SpeechSDK.framework**.

Subscribe to Speech API and get a free trial subscription key

Before creating the example, you must subscribe to Speech API which is part of Cognitive Services. For subscription and key management details, see Subscriptions. Both the primary and secondary key can be used in this tutorial.

Step 1: Install the Example Application and Create the Application Framework

Open Xcode IDE. You have two options, building the example application or building your own application.

If you want build and run the **example application**, the project is embedded here at **samples\SpeechRecognitionServerExample** and can be opened in XCode.

- You will need to paste your subscription key into the file "**settings.plist**" which can be found in the **samples** folder under **SpeechRecognitionServerExample**. (You may ignore the LUIS values if you don't want to use "Intent" right now.)

If you want to **build your own application**, continue on with these instructions.

1. Create a new application project.
2. With the items you downloaded from the SDK, do the following:

   a) Click on the project in the file navigator on the left. Then click on the project or target in the editor that appears. Click on "**Build Settings**", then change from "**Basic**" to "**All**".

   b) Inside the directory to which you unpacked the SDK, you will see the directory, **SpeechSDK/SpeechSDK.framework/Headers**. Add an "**Include Search Path**" to include the **Headers** directory.

c) Inside the directory to which you unpacked the SDK, you will see the directory, **SpeechSDK**. Add a "**Framework Search Path**" to include the **SpeechSDK** directory.

d) Click on the project in the file navigator on the left. Then click on the project or target in the editor that appears. Click on "**General**".

e) Inside the directory to which you unpacked the SDK, you will see the directory, **SpeechSDK/SpeechSDK.framework**. Add **SpeechSDK/SpeechSDK.framework** as a "**Linked Frameworks and Libraries**" via the "**Add Other...**" button found after you click on "**+**".

f) Also add **SpeechSDK.framework** as an "**Embedded Binary**" framework.

g) Note that inside the directory to which you unpacked the SDK in directory **SpeechSDK\Samples\SpeechRecognitionServerExample** there is a XCode buildable example so you can see these settings in action.

Step 2: Build the Application / Example Code

Open ViewController.mm in a new window or find **ViewController.mm** in the downloaded file under **samples\SpeechRecognitionServiceExample**. You will need the **Speech API primary subscription key**. The below code snippet shows where to use the key. (You may ignore the LUIS values if you don't want to use "Intent" right now.)

```
{
    NSString* language = @"en-us";

    NSString* path = [[NSBundle mainBundle] pathForResource:@"settings" ofType:@"plist"];
    NSDictionary* settings = [[NSDictionary alloc] initWithContentsOfFile:path];

    NSString* primaryOrSecondaryKey = [settings objectForKey:(@"primaryKey")];
    NSString* luisAppID = [settings objectForKey:(@"luisAppID")];
    NSString* luisSubscriptionID = [settings objectForKey:(@"luisSubscriptionID")];

    if (isMicrophoneReco) {
        if (!isIntent) {
            micClient = [SpeechRecognitionServiceFactory createMicrophoneClient:(recoMode)
                                                    withLanguage:(language)
                                                        withKey:(primaryOrSecondaryKey)
                                                    withProtocol:(self)];
        }
        else {
            MicrophoneRecognitionClientWithIntent* micIntentClient;
            micIntentClient = [SpeechRecognitionServiceFactory createMicrophoneClientWithIntent:(language)
                                                                withKey:(primaryOrSecondaryKey)
                                                            withLUISAppID:(luisAppID)
                                                            withLUISSecret:(luisSubscriptionID)
                                                             withProtocol:(self)];
            micClient = micIntentClient;
        }
    }
    else {
        if (!isIntent) {
            dataClient = [SpeechRecognitionServiceFactory createDataClient:(recoMode)
                                                withLanguage:(language)
                                                    withKey:(primaryOrSecondaryKey)
                                                withProtocol:(self)];
        }
        else {
            DataRecognitionClientWithIntent* dataIntentClient;
            dataIntentClient = [SpeechRecognitionServiceFactory createDataClientWithIntent:(language)
                                                            withKey:(primaryOrSecondaryKey)
                                                        withLUISAppID:(luisAppID)
                                                        withLUISSecret:(luisSubscriptionID)
                                                         withProtocol:(self)];
            dataClient = dataIntentClient;
        }
    }
}
```

Create a Client

Once your **primaryKey** has been pasted into the example, you can use the **SpeechRecognitionServiceFactory** to create a client of your liking. For example, you can create a client consisting of:

- **DataRecognitionClient:** Speech recognition with PCM data (for example from a file or audio source). The data is broken up into buffers and each buffer is sent to the Speech Recognition Service. No modification is done to the buffers, so the user can apply their own Silence Detection if desired. If the data is provided from wave files, you can send data from the file right to the server. If you have raw data, for example audio coming over Bluetooth, then you first send a format header to the server followed by the data.
- **MicrophoneRecognitionClient:** Speech recognition with audio coming from the microphone. Make sure the microphone is turned on and data from the microphone is sent to the Speech Recognition Service. A built-in "Silence Detector" is applied to the microphone data before it is sent to the recognition service.
- **"WithIntent" Clients:** Use "WithIntent" if you want the server to return additional structured information about the speech to be used by apps to parse the intent of the speaker and drive further actions by the app. To use Intent, you will need to train a model and get an AppID and a Secret. See project LUIS for details.

Select a Language

When you use the SpeechRecognitionServiceFactory to create the Client, you must select a language.

Supported locales include:

LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY
de-DE	zh-TW	zh-HK	ru-RU
es-ES	ja-JP	ar-EG*	da-DK
en-GB	en-IN	fi-FI	nl-NL
en-US	pt-BR	pt-PT	ca-ES
fr-FR	ko-KR	en-NZ	nb-NO
it-IT	fr-CA	pl-PL	es-MX
zh-CN	en-AU	en-CA	sv-SE

*ar-EG supports Modern Standard Arabic (MSA)

Select a Recognition Mode

You also need to provide the recognition mode.

- **ShortPhrase mode:** An utterance up to 15 seconds long. As data is sent to the service, the client will receive multiple partial results and one final multiple n-best choice result.
- **LongDictation mode:** An utterance up to 2 minutes long. As data is sent to the service, the client will receive multiple partial results and multiple final results, based on where the server identifies sentence pauses.

Attach Event Handlers

You can attach various event handlers to the client you created.

- **Partial Results Events:** This event gets called every time the Speech Recognition Service predicts what you might be saying – even before you finish speaking (if you are using the Microphone Client) or have finished sending data (if you are using the Data Client).
- **Error Events:** Called when the service detects an Error.
- **Intent Events:** Called on "WithIntent" clients (only in ShortPhrase mode) after the final reco result has been parsed into a structured JSON intent.
- **Result Events:**
    - **In ShortPhrase mode**, this event is called and returns n-best results after you finish speaking.
    - **In LongDictation mode**, the event handler is called multiple times, based on where the service identifies sentence pauses.
    - **For each of the n-best choices**, a confidence value and a few different forms of the recognized text are returned:
        - **LexicalForm:** This form is optimal for use by applications that need the raw, unprocessed speech recognition result.
        - **DisplayText:** The recognized phrase with inverse text normalization, capitalization, punctuation and profanity masking applied. Profanity is masked with asterisks after the initial

character, for example "d***". This form is optimal for use by applications that display the speech recognition results to users.

- **Inverse Text Normalization (ITN):** An example of ITN is converting result text from "go to fourth street" to "go to 4th St". This form is optimal for use by applications that display the speech recognition results to users.
- **InverseTextNormalizationResult:** Inverse text normalization (ITN) converts phrases like "one two three four" to a normalized form such as "1234". Another example is converting result text from "go to fourth street" to "go to 4th St". This form is optimal for use by applications that interpret the speech recognition results as commands or perform queries based on the recognized text.
- **MaskedInverseTextNormalizationResult:** The recognized phrase with inverse text normalization and profanity masking applied, but no capitalization or punctuation. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to users. Inverse Text Normalization (ITN) has also been applied. An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that use the unmasked ITN results, but also need to display the command or query to users.

Step 3: Run the Example Application

Run the application with the chosen clients, recognition modes and event handlers.

Related Topics

- Get started with Bing Speech Recognition and/or intent in Java on Android
- Get started with Bing Speech API in JavaScript
- Get started with Bing Speech API in cURL

Get Started with Bing Speech Recognition in Java on Android

6/27/2017 • 6 min to read • Edit Online

With Bing Speech Recognition API you can develop Android applications that leverage Microsoft cloud servers to convert spoken audio to text. The API supports real-time streaming, so your application can simultaneously and asynchronously receive partial recognition results at the same time it is sending audio to the service

This article uses a sample application to demonstrate how to use the Bing Speech Recognition API Client Library for Android to develop speech to text applications in Java for Android devices.

Prerequisites

Platform Requirements

The below example has been developed for Android Studio for Windows in Java.

Get the Client Library and Example Application

Download Speech Recognition API Client Library for Android from this link. The downloaded files need to be saved to a folder of your choice. Inside there is both a fully buildable example and the SDK library. The buildable example can be found under **samples** in the **SpeechRecoExample** directory. The two libraries you need to use in your own apps can be found in the **SpeechSDK** folder under **libs** in the **armeabi** and the **x86** folder. The size of **libandroid_platform.so** file is 22 MB but gets reduced to 4MB at deploy time.

Subscribe to Speech API and Get a Free Trial Subscription Key

Before creating the example, you must subscribe to Speech API which is part of Cognitive Services. Click the yellow **"Try for free"** button on one of the offered services, in this case Speech API, and follow the directions. For subscription and key management details, see Subscriptions. Both the primary and secondary key can be used in this tutorial.

Step 1: Install the Example Application and Create the Application Framework

Create an Android application project to implement use of the Speech Recognition API

1. Open Android Studio.Import build.gradle package under samples/SpeechRecoExample.
2. Paste your subscription key into the **primaryKey** string in the **..\samples\SpeechRecoExample\res\values** folder.

   > **NOTE**
   > You don't have to worry about the LUIS values if you don't want to use Intent at this point.)

3. Create a new application project.
4. Using files downloaded from the **speech_SpeechToText-SDK-Android** zip package, do the following:
   a. Copy the **speechsdk.jar** file, found in the **SpeechSDK** folder inside the **Bin** folder, to the "**your-application\app\libs**" folder.
   b. Right click "**app**" in the project tree, select "**Open module settings**", select the "**Dependencies**" tab, and click "**+**" to add a "**File dependency**".
   c. Select the **libs\speechsdk.jar** in the "**Select Path**" dialog box.

d. Copy the **libandroid_platform.so** file to the "**your-application\app\src\main\jniLibs\armeabi**" folder.

You can now run the example application or continue with thes following instructions to build your own application.

Step 2: Build the Example Application

Open MainActivity.java or locate the **MainActivity.java** file within the **samples**, **SpeechRecoExample**, **src**, **com**, **microsoft**, **AzureIntelligentServicesExample** folder from the downloaded **speech_SpeechToText-SDK-Android** zip package. You will need the subscription key you generated above. Once you have added your subscription key to the application, notice that you use the **SpeechRecognitionServiceFactory** to create a client of your liking.

```
void initializeRecoClient()
  {
    String language = "en-us";

    String subscriptionKey = this.getString(R.string.subscription_key);
    String luisAppID = this.getString(R.string.luisAppID);
    String luisSubscriptionID = this.getString(R.string.luisSubscriptionID);

    if (m_isMicrophoneReco && null == m_micClient) {
      if (!m_isIntent) {
        m_micClient = SpeechRecognitionServiceFactory.createMicrophoneClient(this,
                                                m_recoMode,
                                                language,
                                                this,
                                                subscriptionKey);
      }
      else {
        MicrophoneRecognitionClientWithIntent intentMicClient;
        intentMicClient = SpeechRecognitionServiceFactory.createMicrophoneClientWithIntent(this,
                                                    language,
                                                    this,
                                                    subscriptionKey,
                                                    luisAppID,
                                                    luisSubscriptionID);
        m_micClient = intentMicClient;

      }
    }
    else if (!m_isMicrophoneReco && null == m_dataClient) {
      if (!m_isIntent) {
        m_dataClient = SpeechRecognitionServiceFactory.createDataClient(this,
                                              m_recoMode,
                                              language,
                                              this,
                                              subscriptionKey);
      }
      else {
        DataRecognitionClientWithIntent intentDataClient;
        intentDataClient = SpeechRecognitionServiceFactory.createDataClientWithIntent(this,
                                                  language,
                                                  this,
                                                  subscriptionKey,
                                                  luisAppID,
                                                  luisSubscriptionID);
        m_dataClient = intentDataClient;
      }
    }
  }
```

Create a Client:

Create one of the following clients

- **DataRecognitionClient:** Speech recognition with PCM data, for example from a file or audio source. The data is broken up into buffers and each buffer is sent to the Speech Recognition Service. No modification is done to the buffers, so the user can apply their own Silence Detection if desired. If the data is provided from wave files, you can send data from the file directly to the server. If you have raw data, for example audio coming over Bluetooth, then you first send a format header to the server followed by the data.
- **MicrophoneRecognitionClient:** Speech recognition with audio coming from the microphone. Make sure the microphone is turned on and data from the microphone is sent to the Speech Recognition Service. A built-in Silence Detector is applied to the microphone data before it is sent to the recognition service.
- **"WithIntent" clients:** Use "WithIntent" if you want the server to return additional structured information about the speech to be used by apps to parse the intent of the speaker and drive further actions by the app. To use Intent, you will need to train a model and get an AppID and a Secret. See project LUIS for details.

Select a Locale

When you use the SpeechRecognitionServiceFactory to create the Client, you must select a language.

Supported locales include:

LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY	LANGUAGE-COUNTRY
de-DE	zh-TW	zh-HK	ru-RU
es-ES	ja-JP	ar-EG*	da-DK
en-GB	en-IN	fi-FI	nl-NL
en-US	pt-BR	pt-PT	ca-ES
fr-FR	ko-KR	en-NZ	nb-NO
it-IT	fr-CA	pl-PL	es-MX
zh-CN	en-AU	en-CA	sv-SE

*ar-EG supports Modern Standard Arabic (MSA)

Select a Recognition Mode

You also need to provide the recognition mode.

- **ShortPhrase mode:** An utterance up to 15 seconds long. As data is sent to the service, the client will receive multiple partial results and one final multiple n-best choice result.
- **LongDictation mode:** An utterance up to 2 minutes long. As data is sent to the service, the client will receive multiple partial results and multiple final results, based on where the server identifies sentence pauses.

Attach an Event Handler

From the created client, you can attach various event handlers.

- **Partial Results Events:** This event gets called every time the Speech Recognition Server has an idea of what you might be saying – even before you finish speaking (if you are using the Microphone Client) or have finished sending up data (if you are using the Data Client).
- **Error Events:** Called when the Server Detects Error
- **Intent Events:** Called on WithIntent clients (only in ShortPhrase mode) after the final reco result has been

parsed into a structured JSON intent.

- **Result Events:** When you have finished speaking (in ShortPhrase mode), this event is called. You will be provided with n-best choices for the result. In LongDictation mode, he handler's associated with this event will be called multiple times, based on where the server thinks sentence pauses are.

Select Confidence Value and Text Form

For each if the n-best choices, you get a confidence value and few different forms of the recognized text:

- **LexicalForm:** This form is optimal for use by applications that need the raw, unprocessed speech recognition result.
- **DisplayText:** The recognized phrase with inverse text normalization, capitalization, punctuation and profanity masking applied. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to a user.
- **Inverse Text Normalization (ITN):** has also been applied. An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that display the speech recognition results to a user.
- **InverseTextNormalizationResult:** Inverse text normalization (ITN) converts phrases like "one two three four" to a normalized form such as "1234". Another example is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that interpret the speech recognition results as commands or which perform queries based on the recognized text.
- **MaskedInverseTextNormalizationResult:** The recognized phrase with inverse text normalization and profanity masking applied, but not capitalization or punctuation. Profanity is masked with asterisks after the initial character, e.g. "d***". This form is optimal for use by applications that display the speech recognition results to a user. Inverse Text Normalization (ITN) has also been applied. An example of ITN is converting result text from "go to fourth street" to "go to 4th st". This form is optimal for use by applications that use the unmasked ITN results but also need to display the command or query to the user.

Step 3: Run the Example Application

Run the application with the chosen clients, recognition modes and event handlers.

Related Topics

- Get Started with Bing Speech Recognition in C Sharp for Windows in .NET
- Get Started with Bing Speech Recognition and/or intent in Objective C on iOS
- Get Started with Bing Speech API in JavaScript

Convert speech to text with Microsoft APIs

6/27/2017 • 11 min to read • Edit Online

You can choose to convert speech to text by using either an HTTP-based REST protocol or a WebSocket-based protocol. The protocol that you select depends on the needs of your application.

REST Speech Recognition API

The Microsoft REST Speech Recognition API is an HTTP 1.1 protocol definition for building simple speech applications that perform speech recognition. This API is most suitable for applications where continuous user feedback is not required or for platforms that do not support the IETF WebSocket standard. The REST API has the following characteristics:

- Utterances are limited to a maximum of 15 seconds.
- Partial results are not returned. Only the final phrase result is returned.
- Service end-of-speech detection is not supported. Clients must determine the end of speech.
- A single recognition phrase result is returned to the client only after the client stops writing to the request stream.
- Continuous recognition is not supported.

If these features are important to your application's functionality, use the WebSocket API.

WebSocket Speech Recognition API

The Microsoft WebSocket Speech Recognition API is a service protocol definition that uses a WebSocket for bidirectional communication. With this API, you can build full-featured speech applications that provide a rich user experience.

A JavaScript SDK is available based on this version of protocol. SDKs for additional languages and platforms are in development. If your language or platform does not yet have an SDK, you can create your own implementation based on the protocol documentation.

Endpoints

The API endpoints that are based on user scenarios are highlighted here:

MODE	PATH	EXAMPLE URL
Interactive	/speech/recognition/interactive/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/interactive/cognitiveservices/v1?language=pt-BR
Conversation	/speech/recognition/conversation/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
Dictation	/speech/recognition/dictation/cognitiveservices/v1	https://speech.platform.bing.com/speech/recognition/dictation/cognitiveservices/v1?language=fr-FR

Recognition modes

There are three modes of recognition: interactive, conversation, and dictation. The recognition mode adjusts speech recognition based on how the users are likely to speak. Choose the appropriate recognition mode for your application.

> **NOTE**
> Recognition modes might have different behaviors in the REST protocol than they do in the WebSocket protocol. For example, the REST API does not support continuous recognition, even in conversation or dictation mode.

The Microsoft Speech Service returns only one recognition phrase result for all recognition modes. There is a limit of 15 seconds for any single utterance.

Interactive mode

In *interactive* mode, a user makes short requests and expects the application to perform an action in response.

The following characteristics are typical of interactive mode applications:

- Users know they are speaking to a machine and not to another human.
- Application users know ahead of time what they want to say, based on what they want the application to do.
- Utterances typically last about 2-3 seconds.

Conversation mode

In *conversation* mode, users are engaged in a human-to-human conversation.

The following characteristics are typical of conversation mode applications:

- Users know that they are talking to another person.
- Speech recognition enhances the human conversations by allowing one or both participants to see the spoken text.
- Users do not always plan what they want to say.
- Users frequently use slang and other informal speech.

Dictation mode

In *dictation* mode, users recite longer utterances to the application for further processing.

The following characteristics are typical of dictation mode applications:

- Users know that they are talking to a machine.
- Users are shown the speech recognition text results.
- Users often plan what they want to say and use more formal language.
- Users employ full sentences that last 5-8 seconds.

> **NOTE**
> In dictation and conversation modes, the Microsoft Speech Service does not return partial results. Instead, the service returns stable phrase results after silence boundaries in the audio stream. Microsoft might enhance the speech protocol to improve the user experience in these continuous recognition modes.

Recognition language

The *recognition language* specifies the language that your application user speaks. Specify the *recognition language* with the *language* URL query parameter on the connection. The value of the *language* query parameter

must be one of the languages that are supported by the service, as specified in BCP 47 format.

The Microsoft Speech Service rejects invalid connection requests by displaying an `HTTP 400 Bad Request` response. An invalid request is one that:

- Does not include a *language* query parameter value.
- Includes a *language* query parameter that is incorrectly formatted.
- Includes a *language* query parameter that is not one of the support languages.

You may choose to build an application that supports one or all of the languages that are supported by the service.

Example

In the following example, an application uses *conversation* speech recognition mode for a US English speaker.

```
https://speech.platform.bing.com/speech/recognition/conversation/cognitiveservices/v1?language=en-US
```

Interactive and dictation mode

The Microsoft Speech Recognition API supports the following languages in *interactive* and *dictation* modes.

CODE	LANGUAGE	CODE	LANGUAGE
ar-EG	Arabic (Egypt), modern standard	hi-IN	Hindi (India)
ca-ES	Catalan (Spain)	it-IT	Italian (Italy)
da-DK	Danish (Denmark)	ja-JP	Japanese (Japan)
de-DE	German (Germany)	ko-KR	Korean (Korea)
en-AU	English (Australia)	nb-NO	Norwegian (Bokmål) (Norway)
en-CA	English (Canada)	nl-NL	Dutch (Netherlands)
en-GB	English (United Kingdom)	pl-PL	Polish (Poland)
en-IN	English (India)	pt-BR	Portuguese (Brazil)
en-NZ	English (New Zealand)	pt-PT	Portuguese (Portugal)
en-US	English (United States)	ru-RU	Russian (Russia)
es-ES	Spanish (Spain)	sv-SE	Swedish (Sweden)
es-MX	Spanish (Mexico)	zh-CN	Chinese (Mandarin, simplified)
fi-FI	Finnish (Finland)	zh-HK	Chinese (Hong Kong SAR)
fr-CA	French (Canada)	zh-TW	Chinese (Mandarin, Taiwanese)

CODE	LANGUAGE	CODE	LANGUAGE
fr-FR	French (France)		

Conversation mode

The Microsoft Speech Recognition API supports the following languages in *conversation* modes.

CODE	LANGUAGE	CODE	LANGUAGE
ar-EG	Arabic (Egypt), modern standard	It-IT	Italian (Italy)
de-DE	German (Germany)	ja-JP	Japanese (Japan)
en-US	English (United States)	pt-BR	Portuguese (Brazil)
es-ES	Spanish (Spain)	ru-RU	Russian (Russia)
fr-FR	French (France)	zh-CN	Chinese (Mandarin, simplified)

Output format

The Microsoft Speech Service can return a variety of payload formats of recognition phrase results. All payloads are JSON structures.

You can control the phrase result format by specifying the `format` URL query parameter. By default, the service returns `simple` results.

FORMAT	DESCRIPTION
simple	A simplified phrase result containing the recognition status and the recognized text in display form.
detailed	A recognition status and N-best list of phrase results where each phrase result contains all four recognition forms and a confidence score.

The *detailed* format contains the following four recognition forms.

Lexical form

The lexical form is the recognized text, exactly how it occurred in the utterance and without punctuation or capitalization. For example, the lexical form of the address "1020 Enterprise Way" would be *ten twenty enterprise way*, assuming that it was spoken that way. The lexical form of the sentence "Remind me to buy 5 pencils" is *remind me to buy five pencils*.

The lexical form is most appropriate for applications that need to perform non-standard text normalization. The lexical form is also appropriate for applications that need unprocessed recognition words.

Profanity is never masked in the lexical form.

ITN form

Text normalization is the process of converting text from one form to another "canonical" form. For example, the phone number "555-1212" might be converted to the canonical form *five five five one two one two*. *Inverse* text normalization (ITN) reverses this process, converting the words "five five five one two one two" to the inverted

canonical form *555-1212*. The ITN form of a recognition result does not include capitalization or punctuation.

The ITN form is most appropriate for applications that act on the recognized text. For example, an application that allows a user to speak search terms and then uses these terms in a web query would use the ITN form. Profanity is never masked in the ITN form. To mask profanity, use the *Masked ITN form*.

Masked ITN form

Because profanity is naturally a part of spoken language, the Microsoft Speech Service recognizes such words and phrases when they are spoken. Profanity might not, however, be appropriate for all applications, especially those applications with a restricted, non-adult user audience.

The masked ITN form applies profanity masking to the inverse text normalization form. To mask profanity, set the value of the profanity parameter value to `masked`. When profanity is masked, words that are recognized as part of the language's profanity lexicon are replaced with asterisks. For example: *remind me to buy 5 **** pencils*. The masked ITN form of a recognition result does not include capitalization or punctuation.

> **NOTE**
> If the profanity query parameter value is set to `raw`, the masked ITN form is the same as the ITN form. Profanity is *not* masked.

Display form

Punctuation and capitalization signal where to put emphasis, where to pause, and so on, which makes text easier to understand. The display form adds punctuation and capitalization to recognition results, making it the most appropriate form for applications that display the spoken text.

Because the display form extends the masked ITN form, you can set the profanity parameter value to `masked` or `raw`. If the value is set to `raw`, the recognition results include any profanity spoken by the user. If the value is set to `masked`, words recognized as part of the language's profanity lexicon are replaced with asterisks.

Sample responses

All payloads are JSON structures.

The payload format of the `simple` phrase result:

```
{
 "RecognitionStatus": "Success",
 "DisplayText": "Remind me to buy 5 pencils.",
 "Offset": "1236645672289",
 "Duration": "1236645672289"
}
```

The payload format of the `detailed` phrase result:

```
{
 "RecognitionStatus": "Success",
 "Offset": "1236645672289",
 "Duration": "1236645672289",
 "N-Best": [
    {
     "Confidence" : "0.87",
     "Lexical" : "remind me to buy five pencils",
     "ITN" : "remind me to buy 5 pencils",
     "MaskedITN" : "remind me to buy 5 pencils",
     "Display" : "Remind me to buy 5 pencils.",
    },
    {
     "Confidence" : "0.54",
     "Lexical" : "rewind me to buy five pencils",
     "ITN" : "rewind me to buy 5 pencils",
     "MaskedITN" : "rewind me to buy 5 pencils",
     "Display" : "Rewind me to buy 5 pencils.",
    }
 ]
}
```


N-best values

Listeners, whether human or machine, can never be certain that they heard *exactly* what was spoken. A listener can assign a *probability* only to a particular interpretation of an utterance.

In normal conditions, when speaking to others with whom they frequently interact, people have a high probability of recognizing the words that were spoken. Machine-based speech listeners strive to achieve similar accuracy levels and, under the right conditions, they achieve parity with humans.

The algorithms that are used in speech recognition explore alternative interpretations of an utterance as part of normal processing. Usually, these alternatives are discarded as the evidence in favor of a single interpretation becomes overwhelming. In less than optimal conditions, however, the speech recognizer finishes with a list of alternate possible interpretations. The top *N* alternatives in this list are called the *N-best list*. Each alternative is assigned a confidence score. Confidence scores range from 0 to 1. A score of 1 represents the highest level of confidence. A score of 0 represents the lowest level of confidence.

> **NOTE**
> The number of entries in the N-best list vary across multiple utterances. The number of entries can vary across multiple recognitions of the *same* utterance. This variation is a natural and expected outcome of the probabilistic nature of the speech recognition algorithm.

Confidence scores

Confidence scores are integral to speech recognition systems. The Microsoft Speech Service obtains confidence scores from a *confidence classifier*. Microsoft trains the confidence classifier over a set of features that are designed to maximally discriminate between correct and incorrect recognition. Confidence scores are evaluated for individual words and entire utterances.

If you choose to use the confidence scores that are returned by the service, be aware of the following behavior:

- Confidence scores can be compared only within the same recognition mode and language. Do not compare scores between different languages or different recognition modes. For example, a confidence score in interactive recognition mode has *no* correlation to a confidence score in dictation mode.
- Confidence scores are best used on a restricted set of utterances. There is naturally a great degree of variability

in the scores for a large set of utterances.

If you choose to use a confidence score value as a *threshold* on which your application acts, use speech recognition to establish the threshold values.

- Execute speech recognition on a representative sample of utterances for your application.
- Collect the confidence scores for each recognition in the sample set.
- Base your threshold value on some percentile of confidence for that sample.

No single threshold value is appropriate for all applications. An acceptable confidence score for one application might be unacceptable for another application.

Profanity-handling in speech recognition

The Microsoft Speech Service recognizes all forms of human speech, including words and phrases that many people would classify as "profanity." You can control how the service handles profanity by using the *profanity* query parameter. By default, the service masks profanity in *speech.phrase* results and does not return *speech.hypothesis* messages that contain profanity.

***PROFANITY* VALUE**	**DESCRIPTION**
masked	Masks profanity with asterisks. This behavior is the default.
removed	Removes profanity from all results.
raw	Recognizes and returns profanity in all results.

Masked profanity parameter value

To mask profanity, set the *profanity* query parameter to the value *masked*. When the *profanity* query parameter has this value or is not specified for a request, the service *masks* profanity. The service performs masking by replacing profanity in the recognition results with asterisks. When you specify profanity-masking handling, the service does not return *speech.hypothesis* messages that contain profanity.

Removed profanity parameter value

When the *profanity* query parameter has the value *removed*, the service removes profanity from both *speech.phrase* and *speech.hypothesis* messages. The results are the same *as if the profanity words were not spoken*.

Profanity-only utterances

A user might speak *only* profanity when an application has configured the service to remove profanity. For this scenario, if the recognition mode is *dictation* or *conversation*, the service does not return a *speech.result*. If the recognition mode is *interactive*, the service returns a *speech.result* with the status code *NoMatch*.

Raw profanity parameter value

When the *profanity* query parameter has the value *raw*, the service does not remove or mask profanity in either the *speech.phrase* or *speech.hypothesis* messages.

Bing Text to Speech API

7/28/2017 • 9 min to read • Edit Online

Introduction

With the Bing Text to Speech API, your application can send HTTP requests to a cloud server, where text is instantly synthesized into human-sounding speech and returned as an audio file. This API can be used in many different contexts to provide real-time text-to-speech conversion in a variety of different voices and languages.

Voice synthesis request

Authorization token

Every voice synthesis request requires a JSON Web Token (JWT) access token. The JWT access token is passed through in the speech request header. The token has an expiry time of 10 minutes. For information about subscribing and obtaining API keys that are used to retrieve valid JWT access tokens, see Get Started for Free.

The API key is passed to the token service. For example:

```
POST https://api.cognitive.microsoft.com/sts/v1.0/issueToken
Content-Length: 0
```

The required header information for token access is as follows.

NAME	FORMAT	DESCRIPTION
Ocp-Apim-Subscription-Key	ASCII	Your subscription key

The token service returns the JWT access token as `text/plain`. Then the JWT is passed as a `Base64 access_token` to the speech endpoint as an authorization header prefixed with the string `Bearer`. For example:

`Authorization: Bearer [Base64 access_token]`

Clients must use the following endpoint to access the text-to-speech service:

`https://speech.platform.bing.com/synthesize`

> **NOTE**
> Until you have acquired an access token with your subscription key as described earlier, this link generates a 403 response error.

HTTP headers

The following table shows the HTTP headers that are used for voice synthesis requests.

HEADER	VALUE	COMMENTS
Content-Type	application/ssml+xml	The input content type.

HEADER	VALUE	COMMENTS
X-Microsoft-OutputFormat	**1.** ssml-16khz-16bit-mono-tts **2.** raw-16khz-16bit-mono-pcm **3.** audio-16khz-16kbps-mono-siren **4.** riff-16khz-16kbps-mono-siren **5.** riff-16khz-16bit-mono-pcm **6.** audio-16khz-128kbitrate-mono-mp3 **7.** audio-16khz-64kbitrate-mono-mp3 **8.** audio-16khz-32kbitrate-mono-mp3	The output audio format.
X-Search-AppId	A GUID (hex only, no dashes)	An ID that uniquely identifies the client application. This can be the store ID for apps. If one is not available, the ID can be user generated for an application.
X-Search-ClientID	A GUID (hex only, no dashes)	An ID that uniquely identifies an application instance for each installation.
User-Agent	Application name	The application name is required and must be fewer than 255 characters.
Authorization	Authorization token	See the Authorization token section.

Input parameters

Requests to the Bing Text to Speech API are made using HTTP POST calls. The headers are specified in the previous section. The body contains Speech Synthesis Markup Language (SSML) input that represents the text to be synthesized. For a description of the markup used to control aspects of speech such as the language and gender of the speaker, see the SSML W3C Specification.

> **NOTE**
> The maximum size of the SSML input that is supported is 1,024 characters, including all tags.

Example: Voice output request

The following is an example of a voice output request:

```
POST /synthesize
HTTP/1.1
Host: speech.platform.bing.com

X-Microsoft-OutputFormat: riff-8khz-8bit-mono-mulaw
Content-Type: application/ssml+xml
Host: speech.platform.bing.com
Content-Length: 197
Authorization: Bearer [Base64 access_token]

<speak version='1.0' xml:lang='en-US'><voice xml:lang='en-US' xml:gender='Female' name='Microsoft Server Speech Text to Speech Voice (en-US, ZiraRUS)'>Microsoft Bing Voice Output API</voice></speak>
```

Voice output response

The Bing Text to Speech API uses HTTP POST to send audio back to the client. The API response contains the audio stream and the codec, and it matches the requested output format. The audio returned for a given request must not exceed 15 seconds.

Example: Successful synthesis response

The following code is an example of a JSON response to a successful voice synthesis request. The comments and formatting of the code are for purposes of this example only and are omitted from the actual response.

```
HTTP/1.1 200 OK
Content-Length: XXX
Content-Type: audio/x-wav

Response audio payload
```

Example: Synthesis failure

The following example code shows a JSON response to a voice-synthesis query failure:

```
HTTP/1.1 400 XML parser error
Content-Type: text/xml
Content-Length: 0
```

Error responses

ERROR	DESCRIPTION
HTTP/400 Bad Request	A required parameter is missing, empty, or null, or the value passed to either a required or optional parameter is invalid. One reason for getting the "invalid" response is passing a string value that is longer than the allowed length. A brief description of the problematic parameter is included.
HTTP/401 Unauthorized	The request is not authorized.
HTTP/413 RequestEntityTooLarge	The SSML input is larger than what is supported.
HTTP/502 BadGateway	There is a network-related problem or a server-side issue.

The following is an example of an error response:

```
HTTP/1.0 400 Bad Request
Content-Length: XXX
Content-Type: text/plain; charset=UTF-8

Voice name not supported
```

Changing voice output via SSML

This section will show examples of changing certain characteristics of generated voice output like speaking rate, pronunciation etc. by using SSML tags.

1. Adding break:
   ```
   <speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">Welcome to use Microsoft Text to speech voice<break time="100ms" /> in cognitive service.</speak>
   ```
2. Change speaking rate:
   ```
   <speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">
   <prosody rate="+30.00%">Welcome to use Microsoft Text to speech voice in cognitive service.</prosody></speak>
   ```
3. Pronunciation :

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US"><sub alias="World Wide Web Consortium">W3C</sub> </speak>• <speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US"><phoneme alphabet="ipa" ph="t&#x259;mei&#x325;&#x27E;ou&#x325;"> tomato </phoneme></speak>
```

4. Change volume:

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">
<prosody volume="+20.00%">Welcome to use Microsoft Text to speech voice in cognitive service.</prosody></speak>
```

5. Change pitch:

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">Welcome to use <prosody pitch="high">Microsoft Text to speech voice </prosody>in cognitive service.</speak>
```

6. Change prosody contour:

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">
<prosody contour="(80%,+20%) (90%,+30%)" >Good morning.</prosody></speak>
```

7. Insert recorded audio* :

```
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis" xmlns:mstts="http://www.w3.org/2001/mstts" xml:lang="en-US">
<mstts:audiosegment data = "aQdcsdfAsf7sfskAAACUSIIFEasaqKcqEqksk10Nr9zZWIGXigsiS18y..."> Welcome to use Microsoft Text to Speech Voice</mstts:audiosegment></speak>
```

8. Please note the audio data has to be 8k or 16k wav filed in the below format : CRC code (CRC-32): 4 bytes (DWORD) Audio format flag: 4 bytes (DWORD) Sample count: 4 bytes (DWORD) Size of binary body: 4 bytes (DWORD) Binary body: n bytes

Sample application

For implementation details, see the Visual C#.NET text-to-speech sample application.

Supported locales and voice fonts

The following table identifies some of the supported locales and related voice fonts.

LOCALE	GENDER	SERVICE NAME MAPPING
ar-EG*	Female	"Microsoft Server Speech Text to Speech Voice (ar-EG, Hoda)"
ar-SA	Male	"Microsoft Server Speech Text to Speech Voice (ar-SA, Naayf)"
ca-ES	Female	"Microsoft Server Speech Text to Speech Voice (ca-ES, HerenaRUS)"
cs-CZ	Male	"Microsoft Server Speech Text to Speech Voice (cs-CZ, Vit)"
da-DK	Female	"Microsoft Server Speech Text to Speech Voice (da-DK, HelleRUS)"
de-AT	Male	"Microsoft Server Speech Text to Speech Voice (de-AT, Michael)"
de-CH	Male	"Microsoft Server Speech Text to Speech Voice (de-CH, Karsten)"
de-DE	Female	"Microsoft Server Speech Text to Speech Voice (de-DE, Hedda) "
de-DE	Female	"Microsoft Server Speech Text to Speech Voice (de-DE, HeddaRUS)"

LOCALE	GENDER	SERVICE NAME MAPPING
de-DE	Male	"Microsoft Server Speech Text to Speech Voice (de-DE, Stefan, Apollo) "
el-GR	Male	"Microsoft Server Speech Text to Speech Voice (el-GR, Stefanos)"
en-AU	Female	"Microsoft Server Speech Text to Speech Voice (en-AU, Catherine) "
en-AU	Female	"Microsoft Server Speech Text to Speech Voice (en-AU, HayleyRUS)"
en-CA	Female	"Microsoft Server Speech Text to Speech Voice (en-CA, Linda)"
en-CA	Female	"Microsoft Server Speech Text to Speech Voice (en-CA, HeatherRUS)"
en-GB	Female	"Microsoft Server Speech Text to Speech Voice (en-GB, Susan, Apollo)"
en-GB	Female	"Microsoft Server Speech Text to Speech Voice (en-GB, HazelRUS)"
en-GB	Male	"Microsoft Server Speech Text to Speech Voice (en-GB, George, Apollo)"
en-IE	Male	"Microsoft Server Speech Text to Speech Voice (en-IE, Shaun)"
en-IN	Female	"Microsoft Server Speech Text to Speech Voice (en-IN, Heera, Apollo)"
en-IN	Female	"Microsoft Server Speech Text to Speech Voice (en-IN, PriyaRUS)"
en-IN	Male	"Microsoft Server Speech Text to Speech Voice (en-IN, Ravi, Apollo) "
en-US	Female	"Microsoft Server Speech Text to Speech Voice (en-US, ZiraRUS)"
en-US	Female	"Microsoft Server Speech Text to Speech Voice (en-US, JessaRUS)"
en-US	Male	"Microsoft Server Speech Text to Speech Voice (en-US, BenjaminRUS)"
es-ES	Female	"Microsoft Server Speech Text to Speech Voice (es-ES, Laura, Apollo)"
es-ES	Female	"Microsoft Server Speech Text to Speech Voice (es-ES, HelenaRUS)"

LOCALE	GENDER	SERVICE NAME MAPPING
es-ES	Male	"Microsoft Server Speech Text to Speech Voice (es-ES, Pablo, Apollo)"
es-MX	Female	"Microsoft Server Speech Text to Speech Voice (es-MX, HildaRUS)"
es-MX	Male	"Microsoft Server Speech Text to Speech Voice (es-MX, Raul, Apollo)"
fi-FI	Female	"Microsoft Server Speech Text to Speech Voice (fi-FI, HeidiRUS)"
fr-CA	Female	"Microsoft Server Speech Text to Speech Voice (fr-CA, Caroline)"
fr-CA	Female	"Microsoft Server Speech Text to Speech Voice (fr-CA, HarmonieRUS)"
fr-CH	Male	"Microsoft Server Speech Text to Speech Voice (fr-CH, Guillaume)"
fr-FR	Female	"Microsoft Server Speech Text to Speech Voice (fr-FR, Julie, Apollo)"
fr-FR	Female	"Microsoft Server Speech Text to Speech Voice (fr-FR, HortenseRUS)"
fr-FR	Male	"Microsoft Server Speech Text to Speech Voice (fr-FR, Paul, Apollo)"
he-IL	Male	"Microsoft Server Speech Text to Speech Voice (he-IL, Asaf)"
hi-IN	Female	"Microsoft Server Speech Text to Speech Voice (hi-IN, Kalpana, Apollo)"
hi-IN	Female	"Microsoft Server Speech Text to Speech Voice (hi-IN, Kalpana)"
hi-IN	Male	"Microsoft Server Speech Text to Speech Voice (hi-IN, Hemant)"
hu-HU	Male	"Microsoft Server Speech Text to Speech Voice (hu-HU, Szabolcs)"
id-ID	Male	"Microsoft Server Speech Text to Speech Voice (id-ID, Andika)"
it-IT	Male	"Microsoft Server Speech Text to Speech Voice (it-IT, Cosimo, Apollo)"
ja-JP	Female	"Microsoft Server Speech Text to Speech Voice (ja-JP, Ayumi, Apollo)"

LOCALE	GENDER	SERVICE NAME MAPPING
ja-JP	Male	"Microsoft Server Speech Text to Speech Voice (ja-JP, Ichiro, Apollo)"
ja-JP	Female	"Microsoft Server Speech Text to Speech Voice (ja-JP, HarukaRUS)"
ja-JP	Female	"Microsoft Server Speech Text to Speech Voice (ja-JP, LuciaRUS)"
ja-JP	Male	"Microsoft Server Speech Text to Speech Voice (ja-JP, EkaterinaRUS)"
ko-KR	Female	"Microsoft Server Speech Text to Speech Voice (ko-KR, HeamiRUS)"
nb-NO	Female	"Microsoft Server Speech Text to Speech Voice (nb-NO, HuldaRUS)"
nl-NL	Female	"Microsoft Server Speech Text to Speech Voice (nl-NL, HannaRUS)"
pl-PL	Female	"Microsoft Server Speech Text to Speech Voice (pl-PL, PaulinaRUS)"
pt-BR	Female	"Microsoft Server Speech Text to Speech Voice (pt-BR, HeloisaRUS)"
pt-BR	Male	"Microsoft Server Speech Text to Speech Voice (pt-BR, Daniel, Apollo)"
pt-PT	Female	"Microsoft Server Speech Text to Speech Voice (pt-PT, HeliaRUS)"
ro-RO	Male	"Microsoft Server Speech Text to Speech Voice (ro-RO, Andrei)"
ru-RU	Female	"Microsoft Server Speech Text to Speech Voice (ru-RU, Irina, Apollo)"
ru-RU	Male	"Microsoft Server Speech Text to Speech Voice (ru-RU, Pavel, Apollo)"
sk-SK	Male	"Microsoft Server Speech Text to Speech Voice (sk-SK, Filip)"
sv-SE	Female	"Microsoft Server Speech Text to Speech Voice (sv-SE, HedvigRUS)"
th-TH	Male	"Microsoft Server Speech Text to Speech Voice (th-TH, Pattara)"
tr-TR	Female	"Microsoft Server Speech Text to Speech Voice (tr-TR, SedaRUS)"

LOCALE	GENDER	SERVICE NAME MAPPING
zh-CN	Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, HuihuiRUS)"
zh-CN	Female	"Microsoft Server Speech Text to Speech Voice (zh-CN, Yaoyao, Apollo)"
zh-CN	Male	"Microsoft Server Speech Text to Speech Voice (zh-CN, Kangkang, Apollo)"
zh-HK	Female	"Microsoft Server Speech Text to Speech Voice (zh-HK, Tracy, Apollo)"
zh-HK	Female	"Microsoft Server Speech Text to Speech Voice (zh-HK, TracyRUS)"
zh-HK	Male	"Microsoft Server Speech Text to Speech Voice (zh-HK, Danny, Apollo)"
zh-TW	Female	"Microsoft Server Speech Text to Speech Voice (zh-TW, Yating, Apollo)"
zh-TW	Female	"Microsoft Server Speech Text to Speech Voice (zh-TW, HanHanRUS)"
zh-TW	Male	"Microsoft Server Speech Text to Speech Voice (zh-TW, Zhiwei, Apollo)"

*ar-EG supports Modern Standard Arabic (MSA).

Troubleshooting and support

Post all questions and issues to the Bing Speech Service MSDN forum. Include complete details, such as:

- An example of the full request string.
- If applicable, the full output of a failed request, which includes log IDs.
- The percentage of requests that are failing.

Speech Protocol

7/31/2017 • 35 min to read • Edit Online

Microsoft's Speech Service uses a web socket to receive audio and return transcription responses over a single TCP connection. A web socket connection starts out as an HTTP request that contain HTTP headers indicating the client's desire to upgrade the connection to a web socket instead of using HTTP semantics; the server indicates its willingness to participate in the web socket connection by returning an HTTP 101 response. After the exchange of this handshake, both client and service keep the socket open and begin using a message-based protocol to send and receive information.

Connection Protocol

Connections to the Microsoft Speech Service require a web socket that conforms to IETF RFC 6455; the Microsoft Speech Service connection protocol follows that web socket standard specification.

Connections require a web socket handshake. To begin this handshake, the client application sends an HTTPS GET request to the service and includes standard web socket upgrade headers along with other headers that are specific to speech.

```
GET /speech/recognize/interactive/cognitiveservices/v1 HTTP/1.1
Host: speech.platform.bing.com
Upgrade: websocket
Connection: Upgrade
ProtoSec-WebSocket-Key: wPEE5FzwR6mxpsslyRRpgP==
Sec-WebSocket-Version: 13
Authorization: t=EwCIAgALBAAUWkziSCJKS1VkhugDegv7L0eAAJqBYKKTzpPZOeGk7RfZmdBhYY28jl&p=
X-ConnectionId: A140CAF92F71469FA41C72C7B5849253
Origin: https://speech.platform.bing.com
```

The service responds with

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: upgrade
Sec-WebSocket-Key: 2PTTXbeeBXlrrUNsY15n01d/Pcc=
Set-Cookie: SpeechServiceToken=AAAAABAAWTC8ncb8COL; expires=Wed, 17 Aug 2016 15:39:06 GMT; domain=bing.com; path="/"
Date: Wed, 17 Aug 2016 15:03:52 GMT
```

All speech requests require the TLS encryption; the use of unencrypted speech requests is not supported. The following TLS versions are supported:

- TLS 1.2

Connection Identifier

The Microsoft Speech Service requires that all clients include a unique id to identify the connection. Clients **must** include the *X-ConnectionId* header when starting a web socket handshake. The *X-ConnectionId* header value must be a universally unique identifier. Web socket upgrade requests that do not include the *X-ConnectionId*, that do not specify a value for the *X-ConnectionId* header, or that do not include a valid universally unique identifier value will be rejected by the service with a 400 Bad Request response.

Authorization

In addition to the standard web socket handshake headers, speech requests require an *Authorization* header. Connection requests without this header are rejected by the service with a 403 Forbidden HTTP response.

The *Authorization* header must contain a JSON Web Token (JWT) access token.

For information about subscribing and obtaining API keys that are used to retrieve valid JWT access tokens, see Get Started for Free.

The API key is passed to the token service. For example:

```
POST https://api.cognitive.microsoft.com/sts/v1.0/issueToken
Content-Length: 0
```

The required header information for token access is as follows.

NAME	FORMAT	DESCRIPTION
Ocp-Apim-Subscription-Key	ASCII	Your subscription key

The token service returns the JWT access token as `text/plain`. Then the JWT is passed as a `Base64 access_token` to the handshake as an authorization header prefixed with the string `Bearer`. For example:

```
Authorization: Bearer [Base64 access_token]
```

Cookies

Clients **must** support HTTP cookies as specified in RFC 6265.

HTTP Redirection

Clients **must** support the standard redirection mechanisms specified by the HTTP protocol specification.

Speech Endpoint

Clients **must** use a path to the Microsoft Speech Service as outlined in Endpoints.

Reporting Connection Errors

Clients should report all problems and error encountered while making a connection immediately after they next successfully establish a connection. The message protocol for reporting the failed connections is described in the section titled "Connection Failure Telemetry" later in this document.

Connection Duration Limitations

When compared with typical web service HTTP connections, web socket connections last a *long* time. The Microsoft Speech Service does, however, place limitations on the duration of the web socket connections to the service.

- The maximum duration for any active web socket connection is 10 minutes. A connection is active if either the service or the client is sending web socket messages over that connection. The service will terminate the connection without warning once the limit has been reached. Clients should develop user scenarios that do not require the connection to remain active at or near the maximum connection lifetime.
- The maximum duration for any inactive web socket connection is 180 seconds. A connection is inactive if neither the service nor the client has sent any web socket message over the connection. The service will terminate the

inactive web socket connection after the maximum inactive lifetime is reached.

Web Socket Messages

Once a web socket connection is established between the client and the service, both the client and the service may begin sending messages. This section describes the format of these web socket messages.

IETF RFC 6455 specifies that web socket messages can transmit data using either a text or a binary encoding. The two encodings use different on-the-wire formats. Each format is optimized for efficient encoding, transmission and decoding of the message payload.

Text Web Socket Messages

Text web socket messages carry a payload of textual information consisting of a section of headers and a body separated by the familiar double carriage-return-newline pair used for HTTP messages. And, like HTTP messages, text web socket messages specify headers in *name:value* format separated by a single carriage-return-newline pair. Any text included in a text web socket message **must** use UTF-8 encoding.

Text web socket messages must specify a message path in the header *Path*. The value of this header must be one of the speech protocol message types defined later in this document.

Binary Web Socket Messages

Binary web socket messages carry a binary payload. In the Microsoft Speech Service protocol, audio is transmitted to and received from the service using binary web socket messages; all other messages are text web socket messages.

Like text web socket messages, binary web socket messages consist of a header and a body section. The first two bytes of the binary web socket message specify, in big-endian order, the 16-bit integer size of the header section. The minimum header section size is 0 bytes; the maximum size is 8192 bytes. The text in the headers of binary web socket messages **must** use US-ASCII encoding.

Headers in a binary web socket message are encoded in the same format as in text web socket messages, in *name:value* format separated by a single carriage-return-newline pair. Binary web socket messages must specify a message path in the header *Path*. The value of this header must be one of the speech protocol message types defined later in this document.

Speech Protocol

This section describes the web socket message types that make up the protocol for Microsoft Speech Service requests.

Client-Originated Message Types

The following section describes the format and payload of messages that client applications send to the service.

Required Message Headers

The following headers are required for all client-originated messages.

HEADER	VALUE
Path	The message path as specified in this document
X-RequestId	Universally unique identifier in "no-dash" format
X-Timestamp	Client UTC clock timestamp in ISO 8601 format

X-RequestId Header

Client-originated requests are uniquely identified by the *X-RequestId* message header; this header is required for all client-originated messages. The *X-RequestId* header value must be a universally unique identifier in "no-dash" form, for example as *123e4567e89b12d3a456426655440000* but **not** in the canonical form *123e4567-e89b-12d3-a456-426655440000*. Requests without an *X-RequestId* header or with a header value that uses the wrong format for UUIDs will cause the service to terminate the web socket connection.

X-Timestamp Header

Each message sent to the Microsoft Speech Service by a client application **must** include an *X-Timestamp* header. The value for this header should be the time at which the client sends the message. Requests without an *X-Timestamp* header or with a header value that uses the wrong format will cause the service to terminate the web socket connection.

The *X-Timestamp* header value must be of the form 'yyyy'-'MM'-'dd'T'HH':'mm':'ss'.'fffffffZ' where 'fffffff' is a fraction of a second. For example, '12.5' means '12 + 5/10 seconds', '12.526' means '12 plus 526/1000 seconds'. This format complies with ISO 8601 and, unlike the standard HTTP *Date* header, can provide millisecond resolution. Client applications may round timestamps to the nearest millisecond. Client applications should ensure that the device clock accurately tracks time by using a Network Time Protocol (NTP) Server.

Audio

Speech-enabled client applications send audio to the Microsoft Speech Service by converting the audio stream into a series of audio chunks. Each chunk of audio carries a segment of the spoken audio that is to be transcribed by the service. The maximum size of a single audio chunk is 8192 bytes. Audio stream messages are *Binary Web Socket messages*.

Clients use the *audio* message to send an audio chunk to the service. Clients read audio from the microphone in chunks and send these chunks to the Microsoft Speech Service for transcription. The first *audio* message must contain a well-formed header that properly specifies that the audio conforms to one of the encoding formats supported by the service. Additional *audio* messages contain only the binary audio stream data read from the microphone.

Clients may optionally send an *audio* message with a zero-length body; this message tells the service that the client knows that the user has stopped speaking, that the utterance is complete and that the microphone is turned off.

The Microsoft Speech Service uses the first *audio* message that contains a unique request identifier to signal the start of a new request/response cycle or *turn*. After receiving an *audio* message with a new request identifier, the service discards any queued or unsent messages that are associated with any previous turn.

FIELD	DESCRIPTION
Web socket message encoding	Binary
Body	The binary data for the audio chunk. Maximum size is 8192 bytes
Usage	Required

Required Message Headers

The following headers are required for all *audio* messages.

HEADER	VALUE
Path	*audio*
X-RequestId	Universally unique identifier in "no-dash" format
X-Timestamp	Client UTC clock timestamp in ISO 8601 format
Content-Type	The audio content type. Must be one of *audio/x-wav* (PCM) or *audio/silk* (SILK)

Supported Audio Encodings

This section describes the audio codecs supported by the Microsoft Speech Service.

PCM

The Microsoft Speech Service accepts uncompressed pulse code modulation (PCM) audio. Audio is sent to the service in WAV format, so the first audio chunk **must** contain a valid Resource Interchange File Format (RIFF) header. If a client initiates a turn with an audio chunk that does *not* include a valid RIFF header, the service will reject the request and terminate the web socket connection.

PCM audio **must** be sampled at 16 kHz with 16 bits per sample and one channel (*riff-16khz-16bit-mono-pcm*). The Microsoft Speech Service does not support stereo audio streams and will reject audio streams that do not use the specified bit rate, sample rate or number of channels.

OPUS

Opus is a totally open, royalty-free, highly versatile audio codec. The speech service has support for OPUS at a constant bitrate of `32000` or `16000`. Only the `OGG` container for OPUS is currently supported by the service which is specified by the `audio/ogg` mime type. To use OPUS, modify the JavaScript Sample and change the `RecognizerSetup` method to return:

```
return SDK.CreateRecognizerWithCustomAudioSource(
        recognizerConfig,
        authentication,
        new SDK.MicAudioSource(
            new SDK.OpusRecorder(
            {
              mimeType: "audio/ogg",
              bitsPerSecond: 32000
            }
        )
    ));
```

Detecting End of Speech

Since humans do not *explicitly* signal when they are finished speaking, any application that accepts speech as input has two choices for handling the end of speech in an audio stream -- *service end-of-speech detection* and *client end-of-speech detection*. Of these two choices, *service end-of-speech detection* usually provides a better user experience.

Service End-of-Speech Detection

To build the ideal hands-free speech experience, applications should allow the service to detect when the user has finished speaking. Clients send audio from the microphone as *audio* chunks until the service detects silence and responds back with a *speech.endDetected* message.

Client End-of-Speech Detection

Client applications that allow the user to signal the end of speech in some way can also give the service that signal. For example, a client application may have a *stop* or *mute* button that the user can press. To signal end-of-speech, client applications should send an *audio* chunk message with a zero-length body; the Microsoft Speech Service interprets this message as the end of the incoming audio stream.

Speech Config Message

The Microsoft Speech Service needs to know the characteristics of your application to provide the best possible speech recognition. The required characteristics data includes information about the device and operating system that powers your application. You supply this information in the *speech.config* message.

Clients **must** send a *speech.config* message immediately after establishing the connection to the Microsoft Speech Service and before sending any *audio* messages. You only need to send a *speech.config* message *once* per connection.

FIELD	DESCRIPTION
Web socket message encoding	Text
Body	The payload as a JSON structure

Required Message Headers

HEADER NAME	VALUE
Path	*speech.config*
X-Timestamp	Client UTC clock timestamp in ISO 8601 format
Content-Type	application/json; charset=utf-8

As with all client-originated messages in the Microsoft Speech Services protocol, the *speech.config* message **must** include an *X-Timestamp* header that records the client UTC clock time at which the message was sent to the service. The *speech.config* message *does not* require an *X-RequestId* header, since this message is not associated with a particular speech request.

Message Payload

The payload of the *speech.config* message is a JSON structure containing information about the application. An example of this information is shown below. Client and device context information is included in the *context* element of the JSON structure.

```
{
 "context": {
  "system": {
   "version": "2.0.12341",
  },
  "os": {
   "platform": "Linux",
   "name": "Debian",
   "version": "2.14324324"
  },
  "device": {
   "manufacturer": "Contoso",
   "model": "Fabrikan",
   "version": "7.341"
   }
  },
 }
}
```

System Element

The system.version element of the *speech.config* message should contain the version of the speech SDK software used by the client application or device. The value should be in the form *major.minor.build.branch*. The *branch* component may be omitted if it is not applicable.

OS Element

FIELD	DESCRIPTION	USAGE
os.platform	The operating system platform that hosts the application, e.g. Windows, Android, iOS, Linux	Required
os.name	The operating system product name, e.g. Debian, Windows 10	Required
os.version	The version of the operating system, in the form *major.minor.build.branch*	Required

Device Element

FIELD	DESCRIPTION	USAGE
device.manufacturer	The device hardware manufacturer	Required
device.model	The device model	Required
device.version	The device software version provided by the device manufacturer; this value specifies a version of the device that can be tracked by the manufacturer	Required

Telemetry

Client applications *must* acknowledge the end of each turn by sending telemetry about the turn to the Microsoft Speech Service. Turn end acknowledgment allows the Microsoft Speech Service to ensure that all messages necessary for completion of the request and its response have been properly received by the client. Turn end acknowledgment also allows the Microsoft Speech Service to verify that the client applications are performing as

expected; this information will be invaluable if you need help troubleshooting your speech-enabled application.

Clients must acknowledge the end of a turn by sending a *telemetry* message soon after receiving a *turn.end* message. Clients should attempt to acknowledge the *turn.end* as soon as possible; if a client application fails to acknowledge the turn end, the Microsoft Speech Service may terminate the connection with an error. Clients must send only one *telemetry* message for each request and response identified by the *X-RequestId* value.

FIELD	DESCRIPTION
Web socket message encoding	Text
Path	*telemetry*
X-Timestamp	Client UTC clock timestamp in ISO 8601 format
Content-Type	*application/json*
Body	A JSON structure containing client information about the turn

The schema for the body of the *telemetry* message is defined in the section titled "Telemetry Schema" later in this document.

Telemetry for Interrupted Connections

If the network connection fails for any reason during a turn and the client does *not* receive a *turn.end* message from the service, the client should send a *telemetry* message describing the failed request the next time the client makes a connection to the service. Clients do not have to immediately attempt a connection to send the *telemetry* message; the message may be buffered on the client and sent over a future user-requested connection. The *telemetry* message for the failed request **must** use the *X-RequestId* value from the failed request and may be sent to the service as soon as a connection is established, without waiting to send or receive for other messages.

Service-Originated Messages

This section describes the messages that originate in the Microsoft Speech Service and are sent to the client. The Microsoft Speech Service maintains a registry of client capabilities and generates the messages required by each client, so not all clients will receive all messages described here. For brevity, messages are referenced by the value of the *Path* header; for example, we refer to a web socket text message with *Path* value *speech.hypothesis* as a "*speech.hypothesis message*".

Speech Start Detected

The *speech.startDetected* message indicates that the Microsoft Speech Service has detected speech in the audio stream.

FIELD	DESCRIPTION
Web socket message encoding	Text
Path	*speech.startDetected*
Content-Type	application/json; charset=utf-8

FIELD	DESCRIPTION
Body	JSON structure containing information about the conditions at which the start of speech was detected. The *Offset* field in this structure is a media time offset in 100 nanosecond units.

Sample Message

```
Path: speech.startDetected
Content-Type: application/json; charset=utf-8
X-RequestId: 123e4567e89b12d3a456426655440000

{
 "Offset": 100000
}
```

The *Offset* element specifies the offset (in 100-nanosecond units) at which speech was detected in the audio stream, relative to the start of the stream.

Speech Hypothesis

During speech recognition, the Microsoft Speech Service periodically generates hypotheses about the words the service has recognized. The Microsoft Speech Service sends these hypotheses to the client approximately every 300 milliseconds. The *speech.hypothesis* is suitable **only** for enhancing the user speech experience; you must not take any dependency on the content or accuracy of the text in these messages.

FIELD	DESCRIPTION
Web socket message encoding	Text
Path	*speech.hypothesis*
X-RequestId	Universally unique identifier in "no-dash" format
Content-Type	application/json
Body	The speech hypothesis JSON structure
Usage	The *speech.hypothesis* message is applicable to those clients that have some text rendering capability and wish to provide near-real-time feedback of the in-progress recognition to the person who is speaking

Sample Message

```
Path: speech.hypothesis
Content-Type: application/json; charset=utf-8
X-RequestId: 123e4567e89b12d3a456426655440000

{
 "Text": "this is a speech hypothesis",
 "Offset": 0,
 "Duration": 23600000
}
```

The *Offset* element specifies the offset (in 100-nanosecond units) at which the phrase was recognized, relative to the start of the audio stream. The *Duration* element specifies the duration (in 100-nanosecond units) of this speech

phrase.

Clients must not make any assumptions about the frequency, timing or text contained in a speech hypothesis or the consistency of text in any two speech hypotheses. The hypotheses are just snapshots into the transcription process in the service and do not represent a stable accumulation of transcription. For example, a first speech hypothesis may contain the words "fine fun" and the second hypothesis may contain the words "find funny". The Microsoft Speech Service does not perform any post-processing (e.g. capitalization, punctuation) on the text in the speech hypothesis.

Speech Phrase

When the Microsoft Speech Service determines that it has enough information to produce a recognition result that will not change, the service produces a *speech.phrase* message. The Microsoft Speech Service produces these results after it detects that the user has completed a sentence or phrase.

FIELD	DESCRIPTION
Web socket message encoding	Text
Path	*speech.phrase*
Content-Type	application/json
Body	The speech phrase JSON structure

The speech phrase JSON schema includes two fields. The *DisplayText* field represents the recognized phrase after capitalization, punctuation and inverse-text-normalization have been applied and profanity has been masked with asterisks. The *RecognitionStatus* field specifies the status of the recognition.

Sample Message

```
Path: speech.phrase
Content-Type: application/json; charset=utf-8
X-RequestId: 123e4567e89b12d3a456426655440000

{
  "RecognitionStatus": "Success",
  "DisplayText": "Remind me to buy 5 pencils.",
  "Offset": 0,
  "Duration": 12300000
}
```

The values for the RecognitionStatus are given in the table below. The DisplayText field will be present *only* if the RecognitionStatus field has the value *Success*.

STATUS	DESCRIPTION
Success	The recognition was successful and the DisplayText field will be present
NoMatch	Speech was detected in the audio stream, but no words from the target language were matched. See below for more details
InitialSilenceTimeout	The start of the audio stream contained only silence, and the service timed out waiting for speech

STATUS	DESCRIPTION
BabbleTimeout	The start of the audio stream contained only noise, and the service timed out waiting for speech
Error	The recognition service encountered an internal error and could not continue

The *Offset* element specifies the offset (in 100-nanosecond units) at which the phrase was recognized, relative to the start of the audio stream. The *Duration* element specifies the duration (in 100-nanosecond units) of this speech phrase.

NoMatch Recognition Status

The *NoMatch* condition occurs when the Microsoft Speech Service detects speech in the audio stream but is unable to match that speech to the language grammar being used for the request. For example, a *NoMatch* condition might occur if a user says something in German when the recognizer expects US English as the spoken language. The waveform pattern of the utterance would indicate the presence of human speech, but none of the words spoken would match the US English lexicon being used by the recognizer.

Another *NoMatch* condition occurs when the recognition algorithm is unable to find an accurate match for the sounds contained in the audio stream. When this condition happens, the Microsoft Speech Service may produce *speech.hypothesis* messages that contain *hypothesized text* but will produce a *speech.phrase* message in which the *RecognitionStatus* is *NoMatch*. This condition is normal; you must not make any assumptions about the accuracy or fidelity of the text in the *speech.hypothesis* message. Furthermore, you must not assume that because the Microsoft Speech Service produces *speech.hypothesis* messages that the service will be able to produce a *speech.phrase* message with *RecognitionStatus Success*.

Control Messages

The Microsoft Speech Service provides control messages that allow client applications to offer an optimal user experience while still meeting power requirements and regulations. Control messages are identified by the *Path* value.

Speech End Detected

The *speech.endDetected* message specifies that the client application should stop streaming audio to the service.

FIELD	DESCRIPTION
Web socket message encoding	Text
Path	*speech.endDetected*
Body	JSON structure containing the offset at which the end of speech was detected. The offset is represented in 100 nanosecond units offset from the start of audio used for recognition.
Content-Type	application/json; charset=utf-8

Sample Message

```
Path: speech.endDetected
Content-Type: application/json; charset=utf-8
X-RequestId: 123e4567e89b12d3a456426655440000

{
  "Offset": 0
}
```

The *Offset* element specifies the offset (in 100-nanosecond units) at which the phrase was recognized, relative to the start of the audio stream.

Turn Start

The *turn.start* signals the start of a turn from the perspective of the service. The *turn.start* message is the always the **first** response message you will receive for any request. If you do not receive a *turn.start* message, you should assume that the state of the service connection is invalid.

FIELD	DESCRIPTION
Web socket message encoding	Text
Path	*turn.start*
Content-Type	application/json; charset=utf-8
Body	JSON structure

Sample Message

```
Path: turn.start
Content-Type: application/json; charset=utf-8
X-RequestId: 123e4567e89b12d3a456426655440000

{
  "context": {
    "serviceTag": "7B33613B91714B32817815DC89633AFA"
  }
}
```

The body of the *turn.start* message is a JSON structure that contains context for the start of the turn. The *context* element contains a *serviceTag* property; this property specifies a tag value that the service has associated with the turn. This value can be used by Microsoft if you need help troubleshooting failures in your application.

Turn End

The *turn.end* signals the end of a turn from the perspective of the service. The *turn.end* message is the always the **last** response message you will receive for any request. Clients can use the receipt of this message as a signal for cleanup activities and transitioning to an idle state. If you do not receive a *turn.end* message, you should assume that the state of the service connection is invalid; in those cases, you should close the existing connection to the service and reconnect.

FIELD	DESCRIPTION
Web socket message encoding	Text

FIELD	DESCRIPTION
Path	*turn.end*
Body	None

Sample Message

```
Path: turn.end
X-RequestId: 123e4567e89b12d3a456426655440000
```

Telemetry Schema

The body of the *telemetry* message is a JSON structure that contains client information about a turn or an attempted connection. The structure is made up of client timestamps that record when client events occur. Each timestamp must be in ISO 8601 format as described in the section titled *X-Timestamp Header*. *Telemetry* messages that do not specify all the required fields in the JSON structure or which do not use the correct timestamp format may cause the service to terminate the connection to the client. Clients **must** supply valid values for *all required fields*. Clients *should* supply values for optional fields whenever appropriate. The values shown in samples in this section are for illustration only.

Telemetry schema is divided into the following parts: *received message timestamps* and *metrics*. The format and usage of each part is specified below.

Received Message Timestamps

Clients must include time-of-receipt values for *all* messages that it receives after successfully connecting to the service. These values must record the time at which the client *received* each message from the network. The value should not record any other time; for example, the client should not record the time at which it *acted* on the message. The received message timestamps are specified in an array of *name:value* pairs. The name of the pair specifies the *Path* value of the message; the value of the pair specifies the client time when the message was *received*, or, if more than one message of the specified name was received, the value of the pair is an array of timestamps indicating when those messages were received.

```
"ReceivedMessages": [
  { "speech.hypothesis": [ "2016-08-16T15:03:48.172Z", "2016-08-16T15:03:48.331Z", "2016-08-16T15:03:48.881Z" ] },
  { "speech.endDetected": "2016-08-16T15:03:49.721Z" },
  { "speech.phrase": "2016-08-16T15:03:50.001Z" },
  { "turn.end": "2016-08-16T15:03:51.021Z" }
]
```

Clients **must** acknowledge the receipt of **all** messages sent by the service by including timestamps for those messages in the JSON body. If a client fails to acknowledge the receipt of a message, the service may terminate the connection.

Metrics

Clients must include information about events that occurred during the lifetime of a request.

Connection

The *Connection* metric specifies details about connection attempts by the client. The metric must include timestamps of when the web socket connection was started and completed. The *Connection* metric is required **only for the first turn of a connection**; subsequent turns are not required to include this information. If a client makes multiple connection attempts before a connection is established, information about *all* the connection attempts

should be included; for more details, see the section titled "Connection Retries in Telemetry" later in this document.

FIELD	DESCRIPTION	USAGE
Name	Connection	Required
Id	Connection identifier value that used in the *X-ConnectionId* header for this connection request	Required
Start	The time at which the client sent the connection request	Required
End	The time at which the client received notification that the connection was established successfully or, in error cases, rejected, refused or failed	Required
Error	A description of the error that occurred, if any. If the connection was successful, clients should omit this field. The maximum length of this field is 50 characters.	Required for error cases, omitted otherwise

The error description should be at most 50 characters and should ideally be one of the values listed in the table below. If the error condition does not match one of the below values, clients may use a succinct description of the error condition using CamelCasing without whitespace. Note that the ability to send a *telemetry* message requires a connection to the service, so only transient or temporary error conditions can be reported in the *telemetry* message. Error conditions that *permanently* prevent a client from establishing a connection to the service will, of course, prevent the client from sending any message to the service, including *telemetry* messages.

ERROR	USAGE
DNSfailure	The client was unable to connect to the service because of a DNS failure in the network stack
NoNetwork	The client attempted a connection, but the network stack reported that there was no physical network available
NoAuthorization	The client connection failed while attempting to acquire an authorization token for the connection
NoResources	The client ran out of some local resource (e.g. memory) while trying to make a connection
Forbidden	The client was unable to connect to the service because the service returned an HTTP 403 Forbidden status code on the web socket upgrade request
Unauthorized	The client was unable to connect to the service because the service returned an HTTP 401 Unauthorized status code on the web socket upgrade request
BadRequest	The client was unable to connect to the service because the service returned an HTTP 400 Bad Request status code on the web socket upgrade request

ERROR	USAGE
ServerUnavailable	The client was unable to connect to the service because the service returned an HTTP 503 Server Unavailable status code on the web socket upgrade request
ServerError	The client was unable to connect to the service because the service returned an HTTP 500 Internal Error status code on the web socket upgrade request
Timeout	The client's connection request timed out without a response from the service. The End field should contain the time at which the client timed out and stopped waiting for the connection
ClientError	The client terminated the connection because of some internal client error

Microphone

The *Microphone* metric is required for all speech turns. This metric measures the time on the client during which audio input is being actively used for a speech request. The following examples should be used as guidelines for recording *Start* time values for the *Microphone* metric in your client application.

- A client application requires that a user press a physical button to start the microphone; after the button press, the client application reads the input from the microphone and sends it to the Microsoft Speech Service. The *Start* value for the *Microphone* metric records the time after the button push at which the microphone has been initialized and is ready to provide input. The *End* value for the *Microphone* metric records the time at which the client application stopped streaming audio to the service after receiving the *speech.endDetected* message from the service.
- A client application uses a keyword spotter that is "always" listening; only after the keyword spotter detects a spoken trigger phrase does the client application collect the input from the microphone and send it to the Microsoft Speech Service. The *Start* value for the *Microphone* metric records the time at which the keyword spotter notified the client that it should start using input from the microphone. The *End* value for the *Microphone* metric records the time at which the client application stopped streaming audio to the service after receiving the *speech.endDetected* message from the service.
- A client application has access to a constant audio stream and performs silence/speech detection on that audio stream in a *speech detection module*. The *Start* value for the *Microphone* metric records the time that the *speech detection module* notified the client that it should start using input from the audio stream. The *End* value for the *Microphone* metric records the time at which the client application stopped streaming audio to the service after receiving the *speech.endDetected* message from the service.
- A client application is processing the second turn of a multi-turn request and has been informed by a service response message that the microphone should be turned on to gather input for the second turn. The *Start* value for the *Microphone* metric records the time at which the client application has enabled the microphone and has started using input from that audio source. The *End* value for the *Microphone* metric records the time at which the client application stopped streaming audio to the service after receiving the *speech.endDetected* message from the service.

Since the *End* time value for the *Microphone* metric records the time at which the client application stopped streaming audio input, and since in most situations, this event will occur shortly after the client received the *speech.endDetected* message from the service, client applications may verify that they are properly conforming to the protocol by ensuring that the *End* time value for the *Microphone* metric occurs later than the receipt time value for the *speech.endDetected* message. And, since there is usually a delay between the end of one turn and the start of another turn, clients may verify protocol conformance by ensuring that the *Start* time of the *Microphone* metric for any subsequent turn correctly records the time at which the client *started* using the microphone to stream audio

input to the service.

FIELD	DESCRIPTION	USAGE
Name	Microphone	Required
Start	The time at which the client started using audio input from the microphone or other audio stream or received a trigger from the keyword spotter	Required
End	The time at which the client stopped using the microphone or audio stream	Required
Error	A description of the error that occurred, if any. If the microphone operations were successful, clients should omit this field. The maximum length of this field is 50 characters.	Required for error cases, omitted otherwise

Listening Trigger

The *ListeningTrigger* metric measures the time at which the user executes the action that will initiate speech input. The *ListeningTrigger* metric is optional, but clients that can provide this metric are encouraged to do so.

The following examples should be used as guidelines for recording *Start* and *End* time values for the *ListeningTrigger* metric in your client application.

- A client application requires that a user press a physical button to start the microphone; The *Start* value for this metric records the time of the button push. The *End* value records the time at which the button push completes.
- A client application uses a keyword spotter that is "always" listening; after the keyword spotter detects a spoken trigger phrase does the client application read the input from the microphone and send it to the Microsoft Speech Service. The *Start* value for this metric records the time at which the keyword spotter received audio that was then detected as the trigger phrase. The *End* value records the time at which the last word of the trigger phrase was spoken by the user.
- A client application has access to a constant audio stream and performs silence/speech detection on that audio stream in a *speech detection module*. The Start value for this metric records the time that the *speech detection module* received audio that was then detected as speech. The *End* value records the time at which the *speech detection module* detected speech.
- A client application is processing the second turn of a multi-turn request and has been informed by a service response message that the microphone should be turned on to gather input for the second turn. The client application should *not* include a *ListeningTrigger* metric for this turn.

FIELD	DESCRIPTION	USAGE
Name	ListeningTrigger	Optional
Start	The time at which the client listening trigger started	Required
End	The time at which the client listening trigger completed	Required

FIELD	DESCRIPTION	USAGE
Error	A description of the error that occurred, if any. If the trigger operation was successful, clients should omit this field. The maximum length of this field is 50 characters.	Required for error cases, omitted otherwise

Sample Message

```
Path: telemetry
Content-Type: application/json; charset=utf-8
X-RequestId: 123e4567e89b12d3a456426655440000
X-Timestamp: 2016-08-16T15:03:54.183Z

{
 "ReceivedMessages": [
   { "speech.hypothesis": [ "2016-08-16T15:03:48.171Z", "2016-08-16T15:03:48.331Z", "2016-08-16T15:03:48.881Z" ] },
   { "speech.endDetected": "2016-08-16T15:03:49.721Z" },
   { "speech.phrase": "2016-08-16T15:03:50.001Z" },
   { "turn.end": "2016-08-16T15:03:51.021Z" }
 ],
 "Metrics": [
   {
    "Name": "Connection",
    "Id": "A140CAF92F71469FA41C72C7B5849253",
    "Start": "2016-08-16T15:03:47.921Z",
    "End": "2016-08-16T15:03:48.000Z",
   },
   {
    "Name": "ListeningTrigger",
    "Start": "2016-08-16T15:03:48.776Z",
    "End": "2016-08-16T15:03:48.777Z",
   },
   {
    "Name": "Microphone",
    "Start": "2016-08-16T15:03:47.921Z",
    "End": "2016-08-16T15:03:51.921Z",
   },
 ],
}
```

Error Handling

This section describes the kinds of error messages and conditions that your application will need to handle.

HTTP Status Codes

During the web socket upgrade request, the Microsoft Speech Service may return any of the standard HTTP status codes, such as 400 Bad Request, etc. You application must correctly handle these error conditions.

Authorization Errors

The Microsoft Speech Service will return a **403 Forbidden** HTTP status code if incorrect authorization is provided during the web socket upgrade. Among the conditions that can trigger this error code are:

- Missing *Authorization* header
- Invalid authorization token
- Expired authorization token

The 403 Forbidden error does not indicate a problem with the Microsoft Speech Service; this error indicates a

problem with the client application.

Protocol Violation Errors

If the Microsoft Speech Service detects any protocol violations from a client, the service will terminate the web socket connection after returning a **status code** and **reason** for the termination. Client applications can use this information to troubleshoot and fix the violations.

Incorrect Message Format

If a client sends a text or binary message to the service that is not encoded in the correct format given in this specification, the service will close the connection with a *1007 Invalid Payload Data* status code. The service will return this status code for a variety of reasons; examples are:

- Incorrect message format. Binary message has invalid header size prefix. *The client sent a binary message that has an invalid header size prefix*.
- Incorrect message format. Binary message has invalid header size. *The client sent a binary message that specified an invalid header size*.
- Incorrect message format. Binary message headers decoding into UTF-8 failed. *The client sent a binary message containing headers that were not correctly encoded in UTF-8*.
- Incorrect message format. Text message contains no data. *The client sent a text message that contains no body data*.
- Incorrect message format. Text message decoding into UTF-8 failed. *The client sent a text message that was not correctly encoded in UTF-8*.
- Incorrect message format. Text message contains no header separator. *The client sent a text message that did not contain a header separator or used the wrong header separator*.

Missing or Empty Headers

If a client sends a message that does not have the required headers *X-RequestId* or *Path*, the service will close the connection with a *1002 Protocol Error* status code with the reason *Missing/Empty header. {Header name}*.

Request Id Values

If a client sends a message that specifies an *X-RequestId* header with incorrect format, the service will close the connection and return *1002 Protocol Error* status code with reason message *Invalid request. X-RequestId header value was not specified in no-dash UUID format*.

Audio Encoding Errors

If a client sends an audio chunk that initiates a turn and the audio format or encoding does not conform to the required specification, the service will close the connection and return a *1007 Invalid Payload Data* status code with a reason message that indicates the format encoding error source.

Request Id Reuse

Once a turn is complete, if a client sends a message that reuses the request identifier from that turn, the service will close the connection and return a *1002 Protocol Error* status code with the reasons *Invalid request. Reuse of request identifiers is not allowed*.

Connection Failure Telemetry

To ensure the best possible user experience, clients must inform the Microsoft Speech Service of the timestamps for important checkpoints within a connection using the *telemetry* message. It is equally important, however, that clients inform the service of connections that *were attempted but failed*.

For each connection attempt that failed, clients should create a *telemetry* message with a unique *X-RequestId* header value. Since the client was unable to establish a connection, the *ReceivedMessages* field in the JSON body can be omitted; only the *Connection* entry in the *Metrics* field should be included. This entry should include the start

and end timestamps as well as the error condition that was encountered.

Connection Retries in Telemetry

Clients should distinguish *retries* from *multiple connection attempts* by the event that triggers the connection attempt. Connection attempts that are carried out programmatically without any user input are *retries*. Multiple connection attempts that are carried out in response to user input are *multiple connection attempts*. Clients should give each user-triggered connection attempt a unique *X-RequestId* and *telemetry* message; clients should reuse the *X-RequestId* for programmatic retries. If multiple retries were made for a single connection attempt, each retry attempt should be included as a *Connection* entry in the *telemetry* message.

For example, suppose a user speaks the keyword trigger to start a connection and that the first connection attempt fails because of DNS errors but then succeeds on a second attempt that is made programmatically by the client. Since the client retried the connection without requiring additional input from the user, the client should use a single *telemetry* message with multiple *Connection* entries to describe the connection.

As another example, suppose a user speaks the keyword trigger to start a connection and that this connection attempt fails after three retries, at which time the client gives up, stops attempting to connect to the service and informs the user that something went wrong. The user then speaks the keyword trigger again. This time, suppose the client can connect to the service. After connecting, the client should immediately send a *telemetry* message to the service containing three *Connection* entries that describe the connection failures. After receiving the *turn.end* message, the client should send another *telemetry* message that describes the successful connection.

Error Message Reference

HTTP Status Codes

HTTP STATUS CODE	DESCRIPTION	TROUBLESHOOTING
400 Bad Request	The client sent a web socket connection request that was incorrect	Check that you have supplied all the required parameters and HTTP headers and that the values are correct
401 Unauthorized	The client did not include the required authorization information	Check that you are sending the Authorization header in the web socket connection
403 Forbidden	The client sent authorization information, but it was invalid	Check that you are not sending an expired or invalid value in the Authorization header
404 Not Found	The client attempted to access an URL path that is not supported	Check that you are using the correct URL for the web socket connection
500 Server Error	The service encountered an internal error and could not satisfy the request	In most cases, this error will be transient. Retry the request.
503 Service Unavailable	The service was unavailable to handle the request	In most cases, this error will be transient. Retry the request.

Web Socket Error Codes

WEB SOCKET STATUS CODE	DESCRIPTION	TROUBLESHOOTING
1000 Normal Closure	The service closed the web socket connection without an error	If the web socket closure was unexpected, reread the documentation to ensure that you understand how and when the service can terminate the web socket connection.
1002 Protocol Error	The client failed to adhere to protocol requirements	Ensure that you understand the protocol documentation and are clear about the requirements. Read the documentation above about error reasons to see if you are violating protocol requirements.
1007 Invalid Payload Data	The client sent an invalid payload in a protocol message	Check the last message that you sent to the service for errors. Read the documentation above about payload errors.
1011 Server Error	The service encountered an internal error and could not satisfy the request	In most cases, this error will be transient. Retry the request.

Next Steps

Javascript SDK (https://github.com/Azure-Samples/SpeechToText-WebSockets-Javascript)