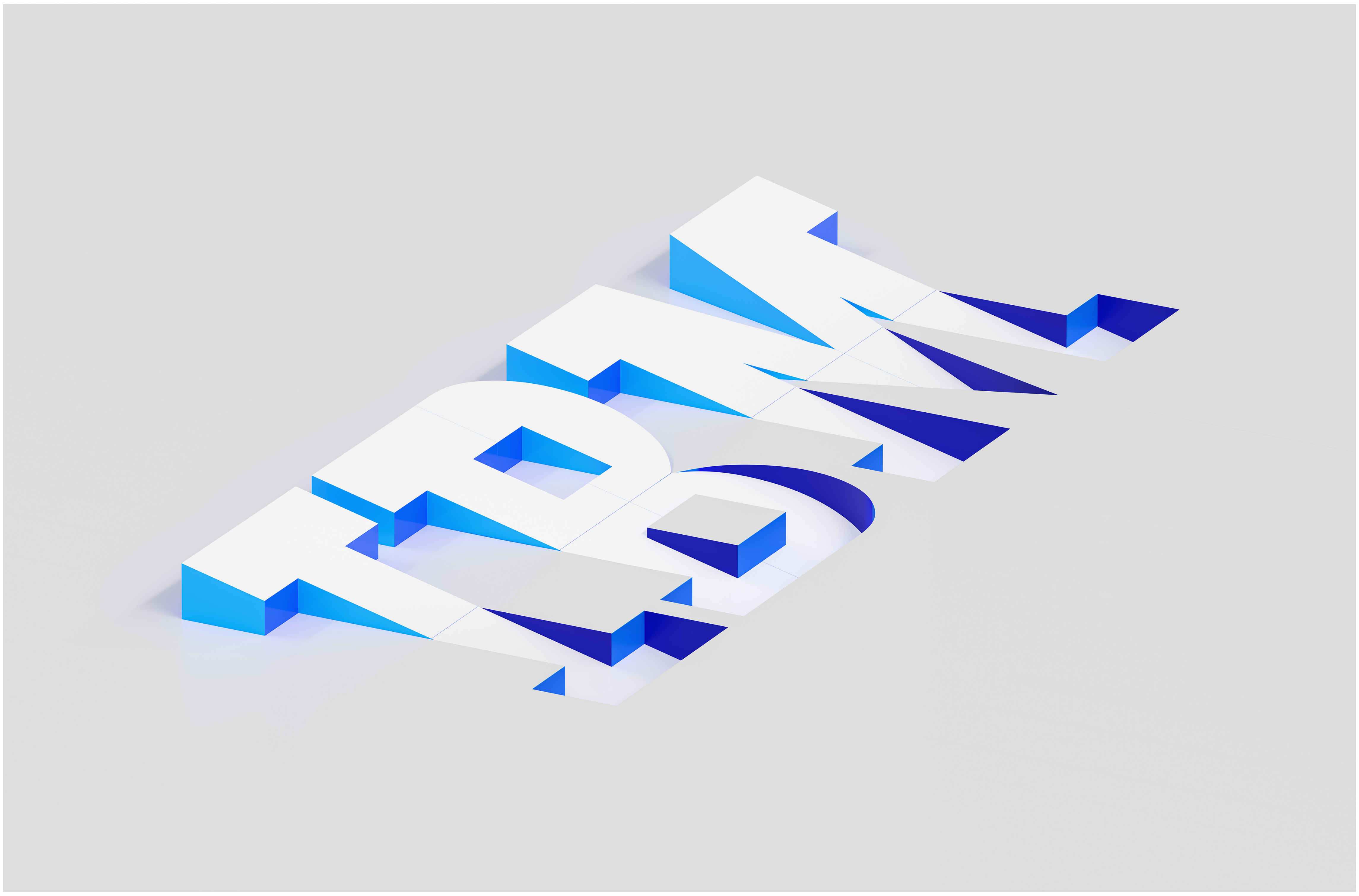
IBM Cloud

Speech to Text

Product guide



Edition notices

This PDF was created on 2024-09-26 as a supplement to *Speech to Text* in the IBM Cloud docs. It might not be a complete set of information or the latest version. For the latest information, see the IBM Cloud documentation at https://cloud.ibm.com/docs/speech-to-text.

Getting started with Speech to Text

The IBM Watson® Speech to Text service transcribes audio to text to enable speech transcription capabilities for applications. This `curl`-based tutorial can help you get started quickly with the service. The examples show you how to call the service's `POST /v1/recognize` method to request a transcript.

Note: The tutorial uses the `curl` command-line utility to demonstrate REST API calls. For more information about `curl`, see Using curl with Watson examples.

IBM Cloud Watch the following video for a visual summary of getting started with the Speech to Text service.

View video: Getting started with the Speech to Text service

Before you begin

IBM Cloud

IBM Cloud

- Create an instance of the service:
  1. Go to the Speech to Text page in the IBM Cloud catalog.
  2. Sign up for a free IBM Cloud account or log in.
  3. Read and agree to the terms of the license agreement.
  4. Click **Create**.
- Copy the credentials to authenticate to your service instance:
  1. View the **Manage** page for the service instance:
     - If you are on the **Getting started** page for your service instance, click the **Manage** entry in the list of topics.
     - If you are on the **Resource list** page, expand the **AI / Machine Learning** grouping in the **Name** column, and click the name of your service instance.
  2. On the **Manage** page, click **Show Credentials** in the **Credentials** box.
  3. Copy the `API Key` and `URL` values for the service instance.

Tip: This tutorial uses an API key to authenticate. In production, use an IAM token. For more information see Authenticating to IBM Cloud.

IBM Cloud Pak for Data

IBM Cloud Pak for Data

The Speech to Text for IBM Cloud Pak for Data must be installed and configured before beginning this tutorial. For more information, see Watson Speech services on Cloud Pak for Data.

1. Create an instance of the service by using the web client, the API, or the command-line interface. For more information about creating a service instance, see Creating a Watson Speech services instance.
2. Follow the instructions in *Creating a Watson Speech services instance* to obtain a Bearer token for the instance. This tutorial uses a Bearer token to authenticate to the service.

Step 1: Transcribe audio with no options

Call the `POST /v1/recognize` method to request a basic transcript of a FLAC audio file with no additional request parameters.

1. Download the sample audio file audio-file.flac.

2. Issue the following command to call the service's `/v1/recognize` method for basic transcription with no parameters. The example uses the `Content-Type` header to indicate the type of the audio, `audio/flac`. The example uses the default language model, `en-US_BroadbandModel`, for transcription.

   IBM Cloud

   - Replace `{apikey}` and `{url}` with your API key and URL.
   - Modify `{path_to_file}` to specify the location of the `audio-file.flac` file.

   ```
   $ curl -X POST -u "apikey:{apikey}" \
   --header "Content-Type: audio/flac" \
   --data-binary @{path_to_file}audio-file.flac \
   "{url}/v1/recognize"
   ```

   IBM Cloud Pak for Data

   - Replace `{token}` and `{url}` with the access token and URL for your service instance.
   - Modify `{path_to_file}` to specify the location of the `audio-file.flac` file.

   ```
   $ curl -X POST \
   --header "Authorization: Bearer {token}" \
   --header "Content-Type: audio/flac" \
   --data-binary @{path_to_file}audio-file.flac \
   "{url}/v1/recognize"
   ```

The service returns the following transcription results:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.96
          "transcript": "several tornadoes touch down as a line of severe thunderstorms swept through Colorado on Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

Step 2: Transcribe audio with options

Call the `POST /v1/recognize` method to transcribe the same FLAC audio file, but specify two transcription parameters.

1. If necessary, download the sample audio file audio-file.flac.

2. Issue the following command to call the service's `/v1/recognize` method with two extra parameters. Set the `timestamps` parameter to `true` to indicate the beginning and end of each word in the audio stream. Set the `max_alternatives` parameter to `3` to receive the three most likely alternatives for the transcription. The example uses the `Content-Type` header to indicate the type of the audio, `audio/flac`, and the request uses the default model, `en-US_BroadbandModel`.

   IBM Cloud

   - Replace `{apikey}` and `{url}` with your API key and URL.
   - Modify `{path_to_file}` to specify the location of the `audio-file.flac` file.

   ```
   $ curl -X POST -u "apikey:{apikey}" \
   --header "Content-Type: audio/flac" \
   --data-binary @{path_to_file}audio-file.flac \
   "{url}/v1/recognize?timestamps=true&max_alternatives=3"
   ```

   IBM Cloud Pak for Data

   - Replace `{token}` and `{url}` with the access token and URL for your service instance.

    - Modify `{path_to_file}` to specify the location of the `audio-file.flac` file.

    ```
    $ curl -X POST \
    --header "Authorization: Bearer {token}" \
    --header "Content-Type: audio/flac" \
    --data-binary @{path_to_file}audio-file.flac \
    "{url}/v1/recognize?timestamps=true&max_alternatives=3"
    ```

The service returns the following results, which include timestamps and three alternative transcriptions:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "timestamps": [
            ["several":, 1.0, 1.51],
            ["tornadoes":, 1.51, 2.15],
            ["touch":, 2.15, 2.5],
            . . .
          ]
        },
        {
          "confidence": 0.96
          "transcript": "several tornadoes touch down as a line of severe thunderstorms swept through Colorado on Sunday "
        },
        {
          "transcript": "several tornadoes touched down as a line of severe thunderstorms swept through Colorado on Sunday "
        },
        {
          "transcript": "several tornadoes touch down as a line of severe thunderstorms swept through Colorado and Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

Next steps

- To try an example application that transcribes text from streaming audio input or from a file that you upload, see the Speech to Text demo.
- For more information about the service's interfaces and features, see Service features.
- For more information about all methods of the service's interfaces, see the API & SDK reference.

About Speech to Text

The IBM Watson® Speech to Text service provides speech transcription capabilities for your applications. The service leverages machine learning to combine knowledge of grammar, language structure, and the composition of audio and voice signals to accurately transcribe the human voice. It continuously updates and refines its transcription as it receives more speech.

The service provides APIs that make it suitable for any application where speech is the input and a textual transcript is the output. It can be used for applications such as voice-automated chatbots, analytic tools for customer-service call centers, and multimedia transcription. Voice control of embedded devices, transcribing meetings and conference calls, and dictating messages and notes are also possible applications, among many others.

The service is ideal for clients who need to extract high-quality speech transcripts from call center audio. Clients in industries such as financial services, healthcare, insurance, and telecommunication can develop cloud-native applications for customer care, customer voice, agent assistance, and other solutions.

Product versions

Speech to Text can be deployed as a managed cloud service or can be installed on premises. This documentation describes how to use both versions of the product. Information such as topics, paragraphs, and examples that applies exclusively to one version is clearly denoted:

- IBM Cloud for managed instances of Speech to Text that are hosted on IBM Cloud or for instances that are hosted on IBM Cloud Pak for Data as a Service.
  - For information about all service updates, see the Release notes for Speech to Text for IBM Cloud.
  - For information about the latest service update, see the 2 May 2023 service update in the release notes.
- IBM Cloud Pak for Data for installed or on-premises instances of Speech to Text for IBM Cloud Pak for Data. For links to information about installing and managing Speech to Text for IBM Cloud Pak for Data, see Installing IBM Watson Speech to Text for IBM Cloud Pak for Data.
  - For information about all service updates, see the Release notes for Speech to Text for IBM Cloud Pak for Data.
  - For information about the latest service update, see the 2 May 2023 (Version 4.6.5) service update in the release notes.

Speech recognition

The Speech to Text service offers three interfaces for speech recognition: a WebSocket interface, a synchronous HTTP interface, and an asynchronous HTTP interface. The interfaces let you specify the language of your audio and its format and sampling rate. They also provide many parameters that you can use to tailor how you request audio and the information that the service sends in response. You can also request metrics about the service's analysis of your audio and the audio itself.

- For more information about the speech recognition interfaces, see Recognizing speech with the service in the service features.
- For more information about the speech recognition parameters, see Using speech recognition parameters in the service features.

Customization

The service provides a customization interface that you can use to tune speech recognition for your language and acoustic requirements. You can expand the vocabulary of a model with domain-specific terminology or adapt a model for the acoustic characteristics of your audio. You can also add grammars to restrict the phrases that the service can recognize. For more information, see Customizing the service in the service features.

Language support

The service supports many languages and dialects:

- Arabic (Modern Standard)
- Chinese (Mandarin)
- Czech
- Dutch (Belgian and Netherlands)
- English (Australian, Indian, United Kingdom, and United States)
- French (Canadian and France)
- German
- Hindi (Indian)
- Italian
- Japanese
- Korean

- Portuguese (Brazilian)
- Spanish (Castilian and Latin American)
- Swedish

For more information about the supported languages and about using large speech models, previous- and next-generation models for speech recognition, see [Using languages and models]().

Audio support

The service accepts audio for transcription in many popular formats:

- Ogg or Web Media (WebM) audio with the Opus or Vorbis codec
- MP3 (or MPEG)
- Waveform Audio File Format (WAV)
- Free Lossless Audio Codec (FLAC)
- Linear 16-bit Pulse-Code Modulation (PCM)
- G.729
- A-Law
- Mu-law (or u-law)
- Basic audio

For more information about the supported audio formats and their characteristics, see Using audio formats.

Integrated use cases

You can use the Speech to Text service with other Watson services to create applications with even greater scope and functionality:

- *AI assistant on the phone* - Eliminate hold times and improve customer satisfaction with IBM® watsonx™ Assistant phone integration. Provide live support to your customers with the pre-built integration of watsonx Assistant, Speech to Text, and IBM Watson® Text to Speech.
- *Analyze customer calls* - Uncover patterns and conduct root-cause analysis on transcriptions of phone calls between your customers and call center agents. Transcribe audio by using Speech to Text, and then analyze the transcription with IBM Watson® Natural Language Understanding.
- *Support agents* - Provide real-time information to improve agent efficiency and focus. Use Speech to Text to transcribe calls live, and then use IBM Watson® Discovery to automatically surface relevant information so that your agent can focus on the customer rather than on the search.

Beta features

IBM occasionally releases features and language support that are classified as beta. Such features are provided so that you can evaluate their functionality. They might be unstable and are subject to change or removal with short notice. They are not intended for use in a production environment.

Beta features might not provide the same level of performance or compatibility as generally available features. Generally available features are ready for use in a production environment.

Pricing

IBM Cloud

The service offers multiple pricing plans to suit your usage and application needs:

- For general information about the pricing plans and answers to common questions, see the Pricing FAQs.
- For more information about the pricing plans or to purchase a plan, see the Speech to Text service in the IBM Cloud® Catalog.

Service features

The IBM Watson® Speech to Text service offers many advanced features to help you get the most from your audio transcription. The service offers multiple speech recognition interfaces, and these interfaces support many features that you can use to manage how you pass your audio to the service and the results that the service returns. You can also customize the service to enhance its vocabulary and to accommodate the acoustic characteristics of your audio. And as with all Watson services, SDKs are available to simplify application development in many programming languages.

Using languages and models

The service supports speech recognition for the many languages listed in Language support. The service provides different models for the languages that it supports. Most language models are generally available (GA) for production use; a few are beta and subject to change.

- For some languages, the service offers large speech models. For more information, see Supported large speech languages and models.
- The service also offers next-generation *Multimedia* and *Telephony* models that improve upon the speech recognition capabilities of the previous-generation models. All next-generation models are GA. Next-generation models return results with greater throughput and higher accuracy than previous-generation models. For more information, see Next-generation languages and models.

For most languages, you can transcribe audio at one of two sampling rates:

- Use *Broadband* or *Multimedia* models for audio that is sampled at a minimum sampling rate of 16 kHz.
- Use *Narrowband* or *Telephony* models for audio that is sampled at a minimum sampling rate of 8 kHz.
- Large speech models supports both audio sampled at sampling rates of either 8 kHz or 16 kHz.

Starting **August 1, 2023**, all previous-generation models are now **discontinued** from the service. New clients must now only use the large speech models or next-generation models. All existing clients must now migrate to the equivalent large speech model or next-generation model. For more information, see Migrating to large speech models.

Using audio formats

The service supports speech recognition for the many audio formats listed in Audio support. Different formats support different sampling rates and other characteristics. By using a format that supports compression, you can maximize the amount of audio data that you can send with a request.

- For more information about understanding audio concepts, see Audio terminology and characteristics.
- For more information about the audio formats that you can use with the service, see Supported audio formats.

Recognizing speech with the service

The Speech to Text service offers a WebSocket interface and synchronous and asynchronous HTTP Representational State Transfer (REST) interfaces.

- The WebSocket interface offers an efficient, low-latency, and high-throughput implementation over a full-duplex connection.
- The synchronous HTTP interface provides a basic interface to transcribe audio with blocking requests.
- The asynchronous HTTP interface provides a non-blocking interface that lets you register a callback URL to receive notifications or poll the service for job status and results.

All interfaces provide the same basic speech recognition capabilities, but you might specify the same parameter as a request header, a query parameter, or a parameter of a JSON object depending on the interface that you use. The service can also return different results depending on the interface and parameters that you use with a request.

- For information about making a speech recognition requests with each of the service's interfaces, see Making a speech recognition request.
- For information about the results of a speech recognition request, see Understanding speech recognition results.

Data limits

The interfaces accept the following maximum amounts of audio data with a single request:

- The WebSocket interface accepts a maximum of 100 MB of audio.
- The synchronous HTTP interface accepts a maximum of 100 MB of audio.
- The asynchronous HTTP interface accepts a maximum of 1 GB of audio.

For more information about using compression to maximize the amount of data that you can send to the service, see Data limits and compression.

Advantages of the WebSocket interface

The WebSocket interface has a number of advantages over the HTTP interface. The WebSocket interface

- Provides a single-socket, full-duplex communication channel. The interface lets the client send multiple requests to the service and receive results over a single connection in an asynchronous fashion.
- Provides a much simpler and more powerful programming experience. The service sends event-driven responses to the client's messages, eliminating the need for the client to poll the server.
- Allows you to establish and use a single authenticated connection indefinitely. The HTTP interfaces require you to authenticate each call to the service.
- Reduces latency. Recognition results arrive faster because the service sends them directly to the client. The HTTP interface requires four distinct requests and connections to achieve the same results.
- Reduces network utilization. The WebSocket protocol is lightweight. It requires only a single connection to perform live-speech recognition.
- Enables audio to be streamed directly from browsers (HTML5 WebSocket clients) to the service.
- Returns results as soon as they are available when you use a large speech model, next-generation model or request interim results.

Using speech recognition parameters

The service's speech recognition interfaces share largely common parameters for transcribing speech to text. The parameters let you tailor aspects of your request, such as whether the data is streamed or sent all at once, and the information that the service includes in its response.

The following sections introduce the speech recognition parameters and their functionality. Some parameters are available only for some speech recognition interfaces or for some languages and models. For information about all parameters and their interface and language support, see the Parameter summary.

Speech or word detection

Use the new parameter speech_begin_event to receive a notification event the moment speech is detected in the audio stream. This feature allows real time applications to learn when you start speaking. A common use case for this feature is implementing barge-in in automated agent systems. Barge-in consists of interrupting audio playback when the caller starts speaking. Set the value to true to make the Speech to Text service send back a speech_begin_event response, which contains the time when speech activity is first detected within the audio stream. You can use this parameter in both standard and low latency mode.

- Parameter name: speech_begin_event
- Request parameter: speech_begin_event = true/false (boolean)
- Response object: "speech_begin_event.begin", for example: {"speech_begin_event": { "begin": }}

Audio transmission and timeouts

- Audio transmission describes how you can pass audio as a continuous stream of data chunks or as a one-shot delivery that passes all of the data at one time. With the WebSocket interface, audio data is always streamed to the service over the connection. With the HTTP interfaces, you can stream the audio or send it all at once.
- Timeouts are used by the service to ensure an active flow of data during audio streaming. When you initiate a streaming session, the service enforces inactivity and session timeouts from which your application must recover gracefully. If a timeout lapses during a streaming session, the service closes the connection.

Interim results and low latency

- Interim results are intermediate hypotheses that the service returns as transcription progresses. They are available only with the WebSocket interface. The service returns final results when a transcript is complete. With the HTTP interfaces, the service always transcribes the entire audio stream before sending any results.

Note: Interim results are not available with large speech models.

- Low latency, when used with certain next-generation models, directs the service to produce final results even more quickly than the models usually do. Low latency is available with the WebSocket and HTTP interfaces. Although low latency further enhances the already improved response times of the models, it might reduce transcription accuracy. When you use the next-generation models with the WebSocket interface, low latency is required to obtain interim results.

Note: Low latency is not available with large speech models.

Speech activity detection

- Speech detector sensitivity adjusts the sensitivity of the service's detection of speech activity. Use the parameter to suppress word insertions from music, coughing, and other non-speech events that can adversely affect the quality of speech recognition.
- Background audio suppression suppresses background audio based on its volume to prevent it from being transcribed as speech. Use the parameter to suppress side conversations or background noise from speech recognition.

Speech audio parsing

- End of phrase silence time specifies the duration of the pause interval at which the service splits a transcript into multiple final results in response to silence. If the service detects pauses or extended silence before it reaches the end of the audio stream, its response can include multiple final results. You can increase or decrease the pause interval to affect the results that you receive.
- Split transcript at phrase end directs the services to split a transcript into multiple final results for semantic features such as sentences. The service bases its understanding of semantic features on the base language model that you use with a request. Custom language models and grammars can also influence how and where the service splits a transcript.

Note: Split transcript at phrase end is not available with large speech models.

- Character insertion bias specifies whether a large speech model or next-generation model is to favor shorter or longer strings as it develops hypotheses during speech recognition. As it develops transcription hypotheses, the service optimizes how it parses audio to balance between competing strings of different lengths. You can indicate that the service is to bias its analysis toward shorter or longer strings.

Note: Character insertion bias is not available with large speech models.

Speaker labels

- Speaker labels identify different speakers from the audio of a multi-participant exchange. The transcription labels the words and times of each speaker's contributions to a multi-participant conversation. Speakers labels are beta functionality.

Keyword spotting and word alternatives

- Keyword spotting identifies spoken phrases that match specified keyword strings with a user-defined level of confidence. Keyword spotting is especially useful when individual phrases from the audio are more important than the full transcription. For example, a customer support system might identify keywords to determine how to route user requests.
- Word alternatives request alternative words that are acoustically similar to the words of a transcript. The words that it identifies must meet a minimum confidence threshold that is specified by the user. The service identifies similar-sounding words and provides their start and end times, as well as its confidence in the possible alternatives.

Note: These features are only supported for previous-generation models. They are not supported for large speech models and next-generation models.

Response formatting and filtering

- Smart formatting version 2 is the new improved feature that converts dates, times, numbers, alphanumerical sequences, currency values, measures, emails, URLs, IP addresses, credit card numbers and dictated punctuations into more readable, conventional forms in final transcripts. This is only supported for large speech models and next generation models in US English, Brazilian Portuguese, French, German, Castilian Spanish, Spanish Latin American, and French Canadian. It is also available for the en-WW_Medical_Telephony model when US English audio is recognized.
- Smart formatting converts dates, times, numbers, currency values, phone numbers, and internet addresses into more readable, conventional forms in final transcripts. For US English, you can also provide keyword phrases to include certain punctuation symbols in final transcripts. Smart formatting is beta functionality.
- Numeric redaction redacts, or masks, numeric data from a final transcript. Redaction is intended to remove sensitive personal information, such as credit card numbers, from final transcripts. Numeric redaction is beta functionality.
- Profanity filtering censors profanity from transcripts and metadata.

Response metadata

- Maximum alternatives provide possible alternative transcripts. The service indicates final results in which it has the greatest confidence.
- Word confidence returns confidence levels for each word of a transcript.
- Word timestamps return timestamps for the start and end of each word of a transcript.

Note: These features are only supported for previous- and next-generation models. They are not supported for large speech models.

Processing and audio metrics

- Processing metrics provide detailed timing information about the service's analysis of the input audio. The service returns the metrics at specified intervals and with transcription events, such as interim and final results. You can use the metrics to gauge the service's progress in transcribing the audio. You can request processing metrics with the WebSocket and asynchronous HTTP interfaces.
- Audio metrics provide detailed information about the signal characteristics of the input audio. The results provide aggregated metrics for the entire input audio at the conclusion of speech processing. You can use the metrics to determine the characteristics and quality of the audio. You can request audio metrics with any of the service's interfaces.

Customizing the service

The customization interface lets you create custom models to improve the service's speech recognition capabilities:

- Custom language models let you define domain-specific words for a base model. Custom language models can expand the service's base vocabulary with terminology specific to domains such as medicine and law. Language model customization is available for large speech models, previous- and next-generation models, though it works differently for the three types of models.
- Custom acoustic models let you adapt a base model for the acoustic characteristics of your environment and speakers. Custom acoustic models improve the service's ability to recognize speech with distinctive acoustic characteristics. Acoustic model customization is available only for previous-generation models.
- Grammars let you restrict the phrases that the service can recognize to those defined in a grammar's rules. By limiting the search space for valid strings, the service can deliver results faster and more accurately. Grammars are created for and used with custom language models. The service generally supports grammars for languages and models for which it supports language model customization. Grammars is available only for previous- and next-generation models.

You can use a custom language model (with or without a grammar), a custom acoustic model, or both for speech recognition with any of the service's interfaces.

- For more information about customization and an overview of its capabilities, see Understanding customization.
- For more information about which languages support customization, see Language support for customization.

Note: IBM Cloud You must have the Plus, Standard, or Premium pricing plan to use language model or acoustic model customization. Users of the Lite plan cannot use the customization interface, but they can upgrade to the Plus plan to gain access to customization. For more information, see the Pricing FAQs.

Using software development kits

SDKs are available for the Speech to Text service to simplify the development of speech applications. The SDKs support many popular programming languages and platforms.

- For a complete list of SDKs and links to the SDKs on GitHub, see Watson SDKs.
- For more information about all methods of the SDKs for the Speech to Text service, see the API & SDK reference.

Learning more about application development

For more information about working with Watson services and IBM Cloud:

- For an introduction, see Getting started with Watson and IBM Cloud.
- For information about using IBM Cloud Identity and Access Management, see Authenticating to Watson services.

Next steps

Explore the features introduced in this topic to gain a more in-depth understanding of the service's capabilities. Each feature includes links to topics that describe it in much greater detail.

- Using languages and models and Using audio formats describe the basic underpinnings of the service's capabilities. You must choose a language and model that are suitable for your audio, and you must understand the characteristics of your audio to make that choice and to pass your audio to the service.
- Recognizing speech with the service provides links to simple examples of speech recognition requests and responses. There are also links to detailed presentations of each of the service's interfaces. Learn more about and experiment with the interfaces to determine which is best suited to your application needs.

- Using speech recognition parameters introduces the many parameters that you can use to tailor speech recognition requests and transcription responses to your needs. The service's WebSocket and HTTP interfaces support an impressive array of capabilities, most of which are common to all supported interfaces. Use the links to find the parameters that are right for you.
- Customizing the service describes the more advanced topics of language model and acoustic model customization, which can help you gain the most from the service's capabilities. The section also presents grammars, which you can use with language models to limit possible responses to precise strings and phrases.
- Using software development kits provide links to the SDKs that are available to simplify application development in many programming languages.
- Learning more about application development provides links to help you get started with Watson services and understand authentication.

Data security

The Speech to Text service provides the following security features to help you protect your user data.

Authenticating to IBM Cloud

IBM Cloud

You authenticate to the service by using IBM Cloud Identity and Access Management (IAM). You can pass either an API key or a bearer token in an `Authorization` header. For more information, see Authenticating to IBM Cloud.

Authenticating to IBM Cloud Pak for Data

IBM Cloud Pak for Data

You authenticate to the service by passing an access token with each request. You pass a bearer token in an `Authorization` header to authenticate. Several methods exist to generate the token, including by using an API key or by username. For more information, see Authenticating to IBM Cloud Pak for Data.

Speech to Text for IBM Cloud Pak for Data is a multi-tenant cloud solution. Your credentials provide access to your data only, and your data is isolated from other users.

Basic data security

The service provides security for all user data both in motion and at rest:

- Transport Layer Security (TLS) 1.2 is used to secure data in transit.
- Advanced Encryption Standard (AES)-256 with Secure Hash Algorithm (SHA)-256 is used to secure data at rest.

For more information about data security for cloud applications, see Security architecture for cloud applications.

Information security

The service supports the European Union General Data Protection Regulation (GDPR) to manage user data. You can pass the `X-Watson-Metadata` header with a request to associate a customer ID with data that the request passes to the service. If necessary, you can then delete the data by using the `DELETE /v1/user_data` method. For more information about these features and their use, see Information security.

IBM Cloud For Premium plans, the service also offers US Health Insurance Portability and Accountability Act (HIPAA) readiness in the Washington, DC, (`us-east`) and Dallas (`us-south`) regions.

Request logging

IBM Cloud

The service lets you control the default request logging that is performed for all Watson services. The service logs request and response data only to improve the service for future users. The logged data is never shared or made public. No data can be exported from the service. For example, users must retain the data that they use for training custom models because it cannot be retrieved from the service.

If you are concerned about the privacy of users' personal information or otherwise do not want your requests to be logged by IBM, you can opt out of the default logging to prevent the service from logging your request and response data. If you opt out, the service logs *no* user data from your requests:

- Results from synchronous HTTP and WebSocket requests are never stored to disk.
- Results from asynchronous HTTP requests are deleted as soon as their time to live expires. For more information, see Asynchronous job retention.

You can choose to opt out of logging at either the account level or the API request level. For more information, see Controlling request logging for Watson services.

Asynchronous job retention

You use the `POST /v1/recognitions` method of the asynchronous HTTP interface to initiate an asynchronous speech recognition job. The service retains the results of each job until its time to live expires. The default time to live for a job is one week, though you can specify a shorter retention period.

You can also use the `DELETE /v1/recognitions/{id}` method to delete the results of a job. You typically delete a job after you obtain its results from the service. Once you delete a job, its results are no longer available. For more information, see Deleting a job.

Data separation and encryption

IBM Cloud

The service's Plus and Premium pricing plans offer different levels of data separation and encryption for users:

- Plus plans are multi-tenant solutions that provide logical separation of data by using common encryption keys.
- Premium plans are single-tenant solutions that provide physical separation of data. Premium plans provide dedicated data storage accounts that use unique encryption keys.

Users of Premium plans can also integrate with IBM® Key Protect for IBM Cloud® to create, import, and manage their encryption keys. This process is commonly referred to as *Bring your own keys* (BYOK). For more information about using IBM® Key Protect for IBM Cloud®, see Protecting sensitive information in your Watson service.

Network endpoints

IBM Cloud

IBM Cloud supports both public and private network endpoints with certain plans. Connections to private network endpoints do not require public internet access. Private network endpoints support routing services over the IBM Cloud private network instead of the public network. A private network endpoint provides a unique IP address that is accessible to you without a VPN connection.

Private network endpoints are supported only for paid plans. Check the plan information for your service to learn about the plans that support private network endpoints. For more information, see Public and private network endpoints.

Virtual private endpoints

IBM Cloud

IBM Cloud® Virtual Private Endpoints for Virtual Private Cloud (VPC) are available with certain plans. Virtual private endpoints enable you to connect to supported IBM Cloud services from your VPC network by using the IP addresses of your choosing, allocated from a subnet within your VPC. Virtual private endpoints are an evolution of the private connectivity to IBM Cloud services. They are virtual IP interfaces that are bound to an endpoint gateway created on a per service or service instance basis.

Virtual private endpoints are supported only for paid plans. Check the plan information for your service to learn about the plans that support virtual private endpoints. For more information, see Virtual Private Endpoints.

CORS support

The service supports Cross-Origin Resource Sharing (CORS). By using CORS, web pages can request resources directly from a foreign domain. CORS circumvents the same-origin security policy, which otherwise prevents such requests. Because the service supports CORS, a web page can communicate directly with the service without passing the request through the web server that hosts the page.

For instance, a web page that is loaded from a server in IBM Cloud can call the customization API directly, bypassing the IBM Cloud server. For more information, see enable-cors.org.

Signed URLs

Signed URLs provide authentication information as a query string. Although signed URLs provide access for a limited time and with limited permissions, they allow any user with such a URL to access the service, regardless of whether that user has an account. The Speech to Text service does not support signed URLs.

FISMA support

IBM Cloud Pak for Data

Federal Information Security Management Act (FISMA) support is available for Speech to Text for IBM Cloud Pak for Data offerings as of version 1.0.1. Speech to Text for IBM Cloud Pak for Data is FISMA High Ready.

FIPS support

IBM Cloud Pak for Data

Speech to Text for IBM Cloud Pak for Data supports running on Federal Information Processing Standard (FIPS)-enabled clusters as of version 4.5.1.

Installing and managing Speech to Text for IBM Cloud Pak for Data

IBM Cloud Pak for Data

To install IBM Watson® Speech to Text for IBM Cloud Pak® for Data, see Watson Speech services on IBM Cloud Pak for Data . That page provides an overview of the Watson Speech services on IBM Cloud Pak for Data and includes links to all installation and management topics for version 4.6.x of the services:

- Preparing to install Watson Speech services
- Installing Watson Speech services
- Post-installation setup for Watson Speech services
- Upgrading Watson Speech services
- Administering Watson Speech services
- Uninstalling Watson Speech services

Installing and managing previous versions

For information about installing and managing previous versions of the Watson Speech services, see the following pages:

- For version 4.5.x, see Watson Speech services on IBM Cloud Pak for Data .
- For version 4.0.x, see Watson Speech to Text on IBM Cloud Pak for Data .

Release notes

Release notes for Speech to Text for IBM Cloud

IBM Cloud

The following features and changes were included for each release and update of managed instances of IBM Watson® Speech to Text that are hosted on IBM Cloud or for instances that are hosted on IBM Cloud Pak for Data as a Service. Unless otherwise noted, all changes are compatible with earlier releases and are automatically and transparently available to all new and existing applications.

For information about known limitations of the service, see Known limitations.

Note: For information about releases and updates of the service for IBM Cloud Pak for Data, see Release notes for Speech to Text for IBM Cloud Pak for Data.

23 August 2024

All Large Speech Models are now generally available

The large speech models for all languages are now generally available (GA). They are supported for use in production environments and applications.

- For more information about large speech models, see Large speech languages and models.
- For more information about the features that are supported for large speech models, see Supported features for large speech models.

18 June 2024

New large speech models for Brazilian Portuguese and Spanish are now in open beta

The large speech models for Brazilian Portuguese and Spanish are now in open beta. Spanish includes the Castilian, Argentinian, Chilean, Colombian, Mexican, and Peruvian dialects.

- For more information about large speech models, see Large speech languages and models.
- For more information about the features that are supported for large speech models, see Supported features for large speech models.

15 May 2024

Large Speech Model for English is now generally available

The large speech model for English, which includes the United States, Australian, Indian, and United Kingdom dialects, is now generally available (GA). It is supported for use in production environments and applications.

- For more information about large speech models, see Large speech languages and models.
- For more information about the features that are supported for large speech models, see Supported features for large speech models.

07 March 2024

Large Speech Model for US English in Open Beta

The new Large speech model for US English is in open beta. See Large speech languages and models for more details with supported features (beta).

30 November 2023

Speech to Text parameter speech_begin_event

This parameter would enable the client application to know that some words or speech is detected and Speech to Text is in the process of decoding. For more details, see Using speech recognition parameters .

Parameter 'mapping_only' for custom words

By using the 'mapping_only' parameter, you can use custom words directly to map 'sounds_like' (or word) to 'display_as' value as post-processing instead of training. For more information, see The words resource.

See the guidelines for Non-Japanese and Japanese.

Support for Brazilian-Portuguese and French-Canadian on new improved next-generation language model customization

Language model customization for Brazilian-Portuguese and French-Canadian next-generation models is recently added. This service update includes further internal improvements.

New Smart Formatting Feature

A new smart formatting feature for next-generation models is supported in US English, Brazilian Portuguese, French and German languages. See Smart formatting Version for details.

Support for Castilian Spanish and LATAM Spanish on new improved next-generation language model customization

The language model customization for Castilian Spanish and LATAM Spanish next-generation models are added. This service update includes further internal improvements.

Large Speech Models for English, Japanese, and French - for early access

For early access feature, Large Speech Models are available for English, Japanese and French languages for you in IBM Watson Speech-to-Text and IBM watsonx Assistant. The feature set for these Large Speech Models is limited, but more accurate than Next-Generation models and are faster and cheaper to run due to smaller size and better streaming mode capability.

If you are interested in testing these base models, and sharing results and feedback, contact our Product Management team by filling out this form.

28 July 2023

Important: All previous-generation models are discontinued starting August 1, 2023

Important: All previous-generation models are now discontinued from the service. New clients must now only use the next-generation models. All existing clients must now migrate to the equivalent next-generation model. For more information about all next-generation models, see Next-generation languages and models. For more information on how to migrate to the next-generation models, see Migrating to next-generation models.

9 June 2023

Defect fix: Creating and training a custom Language Model is now optimal for both standard and low-latency Next-Generation models

Defect fix: When creating and training a custom Language Model with corpora text files and / or custom words using a Next-generation low-latency model, it is now performing the same way as with a standard model. Previously, it was not optimal only when using a Next-Generation low-latency model.

Defect fix: STT Websockets sessions no longer fail due to tensor error message

Defect fix: When using STT websockets, sessions no longer fail due to an error message “STT returns the error: Sizes of tensors must match except in dimension 0”.

18 May 2023

Updates to English next-generation Medical telephony model

The English next-generation Medical telephony model has been updated for improved speech recognition:

- `en-WW_Medical_Telephony`

Added support for French and German on new improved next-generation language model customization

Language model customization for French and German next-generation models was recently added. This service update includes further internal improvements.

For more information about improved next-generation customization, see

- February 27, 2023 service update
- Improved language model customization for next-generation models

Defect fix: Custom words containing half-width Katakana characters now return a clear error message with Japanese Telephony model

Defect fix: Per the documentation, only full-width Katakana characters are accepted in custom words and the next generation models now show an error message to explain that it's not supported. Previously, when creating custom words containing half-width Katakana characters, no error message was provided.

Defect fix: Japanese Telephony language model no longer fails due to long training time

Defect fix: When training a custom language model with Japanese Telephony, the service now effectively handles large numbers of custom words without failing.

2 May 2023

New procedure for upgrading a custom model that is based on an improved next-generation model

Two approaches are now available to upgrade a custom language model to an improved next-generation base model. You can still modify and then retrain the custom model, as already documented. But now, you can also upgrade the custom model by including the query parameter `force=true` with the `POST /v1/customizations/{customization_id}/train` request. The `force` parameter upgrades the custom model regardless of whether it contains changes (is in the `ready` or `available` state).

For more information, see Upgrading a custom language model based on an improved next-generation model.

Guidance for adding words to custom models that are based on improved next-generation models

The documentation now offers more guidance about adding words to custom models that are based on improved next-generation models. For performance reasons during training, the guidance encourages the use of corpora rather than the direct addition of custom words whenever possible.

For more information, see Guidelines for adding words to custom models based on improved next-generation models.

Japanese custom words for custom models that are based on improved next-generation models are handled differently

For Japanese custom models that are based on next-generation models, custom words are handled differently from other languages. For Japanese, you can add a custom word or sounds-like that does not exceed 25 characters in length. If your custom word or sounds-like exceeds that limit, the service adds the word to the custom model as if it were added by a corpus. The word does not appear as a custom word for the model.

For more information, see Guidelines for adding words to Japanese models based on improved next-generation models.

12 April 2023

Defect fix: The WebSocket interface now times out as expected when using next-generation models

Defect fix: When used for speech recognition with next-generation models, the WebSocket interface now times out as expected after long periods of silence. Previously, when used for speech recognition of short audio files, the WebSocket session could fail to time out. When the session failed to time out, the service did not return a final hypothesis to the waiting client application, and the client instead timed while waiting for the results.

6 April 2023

Defect fix: Limits to allow completion of training for next-generation Japanese custom models

Defect fix: Successful training of a next-generation Japanese custom language model requires that custom words and sounds-likes added to the model each contain no more than 25 characters. For the most effective training, it is recommended that custom words and sounds-likes contain no more than 20 characters. Training of Japanese custom models with longer custom words and sounds-likes fails to complete after multiple hours of training.

If you need to add the equivalent of a long word or sounds-like to a next-generation Japanese custom model, take these steps:

1. Add a shorter word or sounds-like that captures the essence of the longer word or sounds-like to the custom model.
2. Add one or more sentences that use the longer word or sounds-like to a corpus.
3. Consider adding sentences to the corpus that provide more context for the word or sounds-like. Greater context gives the service more information with which to recognize the word and apply the correct sounds-like.
4. Add the corpus to the custom model.
5. Retrain the custom model on the combination of the shorter word or sounds-like and the corpus that contains the longer string.

The limits and steps just described allow next-generation Japanese custom models to complete training. Keep in mind that adding large numbers of new custom words to a custom language model increases the training time of the model. But the increased training time occurs only when the custom model is initially trained on the new words. Once the custom model has been trained on the new words, training time returns to normal.

For more information, see

- Add a corpus to the custom language model
- Add words to the custom language model
- Train the custom language model
- Working with corpora and custom words for next-generation models

Further improvements to updated next-generation language model customization

Language model customization for English and Japanese next-generation models was recently improved. This service update includes further internal improvements. For more information about improved next-generation customization, see

- 27 February 2023 service update and
- Improved language model customization for next-generation models

13 March 2023

Defect fix: Smart formatting for US English dates is now correct

Defect fix: Smart formatting now correctly includes days of the week and dates when both are present in the spoken audio, for example, `Tuesday February 28`. Previously, in some cases the day of the week was omitted and the date was presented incorrectly. Note that smart formatting is beta functionality.

Defect fix: Update documentation for speech hesitation words for next-generation models

Defect fix: Documentation for speech hesitation words for next-generation models has been updated. More details are provided about US English and Japanese hesitation words. Next-generation models include the actual hesitation words in transcription results, unlike previous-generation models, which include only hesitation markers. For more information, see Speech hesitations and hesitation markers.

27 February 2023

New Japanese next-generation telephony model

The service now offers a next-generation telephony model for Japanese: `ja-JP_Telephony`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Improved language model customization for next-generation English and Japanese models

The service now provides improved language model customization for next-generation English and Japanese models:

- `en-AU_Multimedia`
- `en-AU_Telephony`
- `en-IN_Telephony`
- `en-GB_Multimedia`
- `en-GB_Telephony`
- `en-US_Multimedia`
- `en-US_Telephony`
- `ja-JP_Multimedia`
- `ja-JP_Telephony`

Visible improvements to the models: The new technology improves the default behavior of the new English and Japanese models. Among other changes, the new technology optimizes the default behavior for the following parameters:

- The default `customization_weight` for custom models that are based on the new versions of these models changes from `0.2` to `0.1`.
- The default `character_insertion_bias` for custom models that are based on the new versions of these models remains `0.0`, but the models have changed in a manner that makes use of the parameter for speech recognition less necessary.

Upgrading to the new models: To take advantage of the improved technology, you must upgrade any custom language models that are based on the new models. To upgrade to the new version of one of these base models, do the following:

1. Change your custom model by adding or modifying a custom word, corpus, or grammar that the model contains. Any change that you make moves the model to the `ready` state.

2. Use the `POST /v1/customizations/{customization_id}/train` method to retrain the model. Retraining upgrades the custom model to the new technology and moves the model to the `available` state.

   Known issue: At this time, you cannot use the `POST /v1/customizations/{customization_id}/upgrade_model` method to upgrade a custom model to one of the new base models. This issue will be addressed in a future release.

Using the new models: Following the upgrade to the new base model, you are advised to evaluate the performance of the upgraded custom model by paying special attention to the `customization_weight` and `character_insertion_bias` parameters for speech recognition. When you retrain your custom model:

- The custom model uses the new default `customization_weight` of `0.1` for your custom model. A non-default `customization_weight` that you had associated with your custom model is removed.
- The custom model might no longer require use of the `character_insertion_bias` parameter for optimal speech recognition.

Improvements to language model customization render these parameters less important for high-quality speech recognition:

- If you use the default values for these parameters, continue to do so after the upgrade. The default values will likely continue to offer the best results for speech recognition.
- If you specify non-default values for these parameters, experiment with the default values following upgrade. Your custom model might work well for speech recognition with the default values.

If you feel that using different values for these parameters might improve speech recognition with your custom model, experiment with incremental changes to determine whether the parameters are needed to improve speech recognition.

Note: At this time, the improvements to language model customization apply only to custom models that are based on the next-generation English or Japanese base language models listed earlier. Over time, the improvements will be made available for other next-generation language models.

More information: For more information about upgrading and about speech recognition with these parameters, see

- Improved language model customization for next-generation models
- Upgrading custom models
- Using customization weight
- Character insertion bias

Defect fix: Grammar files now handle strings of digits correctly

Defect fix: When grammars are used, the service now handles longer strings of digits correctly. Previously, it was failing to complete recognition or returning incorrect results.

15 February 2023

Important: All previous-generation models are deprecated and will reach end of service on 31 July 2023

Important: All previous-generation models are deprecated and will reach end of service effective **31 July 2023**. On that date, all previous-generation models will be removed from the service and the documentation. The previous deprecation date was 3 March 2023. The new date allows users more time to migrate to the appropriate next-generation models. But users must migrate to the equivalent next-generation model by 31 July 2023.

Most previous-generation models were deprecated on 15 March 2022. Previously, the Arabic and Japanese models were not deprecated. Deprecation now applies to *all* previous-generation models.

- For more information about the next-generation models to which you can migrate from each of the deprecated models, see Previous-generation languages and models
- For more information about migrating from previous-generation to next-generation models, see Migrating to next-generation models.
- For more information about all next-generation models, see Next-generation languages and models

Note: When the previous-generation `en-US_BroadbandModel` is removed from service, the next-generation `en-US_Multimedia` model will become the default model for speech recognition requests.

Defect fix: Improved training time for next-generation custom language models

Defect fix: Training time for next-generation custom language models is now significantly improved. Previously, training time took much longer than necessary, as reported for training of Japanese custom language models. The problem was corrected by an internal fix.

Defect fix: Dynamically generated grammar files now work properly

Defect fix: Dynamically generated grammar files now work properly. Previously, dynamic grammar files could cause internal failures, as reported for integration of Speech to Text with IBM® watsonx™ Assistant. The problem was corrected by an internal fix.

20 January 2023

Deprecated Arabic and United Kingdom model names are no longer available

The following Arabic and United Kingdom model names are no longer accepted by the service:

- `ar-AR_BroadbandModel` - Use `ar-MS_BroadbandModel` instead.
- `en-UK_NarrowbandModel` - Use `en-GB_NarrowbandModel` instead.
- `en-UK_BroadbandModel` - Use `en-GB_BroadbandModel` instead.

The Arabic model name was deprecated on 2 December 2020. The UK English model names were deprecated on 14 July 2017.

Cloud Foundry deprecation and migration to resource groups

IBM announced the deprecation of IBM Cloud Foundry on 31 May 2022. As of 30 November 2022, new IBM Cloud Foundry applications cannot be created and only existing users are able to deploy applications. IBM Cloud Foundry reaches end of support on 1 June 2023. At that time, any IBM Cloud Foundry application runtime instances running IBM Cloud Foundry applications will be permanently disabled, deprovisioned, and deleted. For more information about the deprecation, see Deprecation of IBM Cloud Foundry.

To continue to use your IBM Cloud applications beyond 1 June 2023, you must migrate to resource groups before that date. Resource groups are conceptually similar to Cloud Foundry spaces. They include several extra benefits, such as finer-grained access control by using IBM Cloud Identity and Access Management (IAM), the ability to connect service instances to apps and service across different regions, and an easy way to view usage per group. For more information about migration, see Migrating Cloud Foundry service instances and apps to a resource group.

The `max_alternatives` parameter is now available for use with next-generation models

The `max_alternatives` parameter is now available for use with all next-generation models. The parameter is generally available for all next-generation models. For more information, see Maximum alternatives.

Defect fix: Allow use of both `max_alternatives` and `end_of_phrase_silence_time` parameters with next-generation models

Defect fix: When you use both the `max_alternatives` and `end_of_phrase_silence_time` parameters in the same request with next-generation models, the service now returns multiple alternative transcripts while also respecting the indicated pause interval. Previously, use of the two parameters in a single request generated a failure. (Use of the `max_alternatives` parameter with next-generation models was previously available as an experimental feature to a limited number of customers.)

Defect fix: Update French Canadian next-generation telephony model (upgrade required)

Defect fix: The French Canadian next-generation telephony model, `fr-CA_Telephony`, was updated to address an internal inconsistency that could cause an error during speech recognition. *You need to upgrade any custom models that are based on the `fr-CA_Telephony` model.* For more information about upgrading custom models, see

- Upgrading custom models
- Using upgraded custom models for speech recognition

Defect fix: Add documentation guidelines for creating Japanese sounds-likes based on next-generation models

Defect fix: In sounds-likes for Japanese custom language models that are based on next-generation models, the character-sequence `ウー` is ambiguous in some left contexts. Do not use characters (syllables) that end with the phoneme `/o/`, such as `ロ` and `ト`. In such cases, use `ウウ` or just `ウ` instead of `ウー`. For example, use `ロウウマン` or `ロウマン` instead of `ロウーマン`. For more information, see Guidelines for Japanese.

Adding words directly to custom models that are based on next-generation models increases the training time

Adding custom words directly to a custom model that is based on a next-generation model causes training of a model to take a few minutes longer than it otherwise would. If you are training a model with custom words that you added by using the `POST /v1/customizations/{customization_id}/words` or `PUT /v1/customizations/{customization_id}/words/{word_name}` method, allow for some minutes of extra training time for the model. For more information, see

- Add words to the custom language model
- Monitoring the train model request

Maximum hours of audio resources for custom acoustic models in the Tokyo location has been increased

The maximum hours of audio resources that you can add to custom acoustic models in the Tokyo location is again 200 hours. Previously, the maximum was reduced to 50 hours for the Tokyo region. That reduction has been rescinded and postponed until next year. For more more information, see Maximum hours of audio.

5 December 2022

New Netherlands Dutch next-generation multimedia model

The service now offers a next-generation multimedia model for Netherlands Dutch: `nl-NL_Multimedia`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Defect fix: Correct custom word recognition in transcription results for next-generation models

Defect fix: For language model customization with next-generation models, custom words are now recognized and used in all transcripts. Previously, custom words sometimes failed to be recognized and used in transcription results.

Defect fix: Correct use of `display_as` field in transcription results for next-generation models

Defect fix: For language model customization with next-generation models, the value of the `display_as` field for a custom word now appears in all transcripts. Previously, the value of the `word` field sometimes appeared in transcription results.

Defect fix: Update custom model naming documentation

Defect fix: The documentation now provides detailed rules for naming custom language models and custom acoustic models. For more information, see

- Create a custom language model
- Create a custom acoustic model
- API & SDK reference

20 October 2022

Updates to English next-generation telephony models

The English next-generation telephony models have been updated for improved speech recognition:

- `en-AU_Telephony`
- `en-GB_Telephony`
- `en-IN_Telephony`
- `en-US_Telephony`

All of these models continue to support low latency. You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

Defect fix: Update Japanese next-generation multimedia model (upgrade required)

Defect fix: The Japanese next-generation multimedia model, `ja-JP_Multimedia`, was updated to address an internal inconsistency that could cause an error during speech recognition with low latency. *You need to upgrade any custom models that are based on the `ja-JP_Multimedia` model.* For more information about upgrading custom models, see

- Upgrading custom models
- Using upgraded custom models for speech recognition

7 October 2022

New Swedish next-generation telephony model

The service now offers a next-generation telephony model for Swedish: `sv-SE_Telephony`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Updates to English next-generation telephony models

The English next-generation telephony models have been updated for improved speech recognition:

- `en-AU_Telephony`
- `en-GB_Telephony`
- `en-IN_Telephony`
- `en-US_Telephony`

All of these models continue to support low latency. You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

21 September 2022

New Activity Tracker event for GDPR deletion of user information

The service now returns an Activity Tracker event when you use the `DELETE /v1/user_data` method to delete all information about a user. The event is named `speech-to-text.gdpr-user-data.delete`. For more information, see Activity Tracker events.

Defect fix: Update some next-generation models to improve low-latency response time

Defect fix: The following next-generation models were updated to improve their response time when the `low_latency` parameter is used:

- `en-IN_Telephony`
- `hi-IN_Telephony`

- `it-IT_Multimedia`
- `nl-NL_Telephony`

Previously, these models did not return recognition results as quickly as expected when the `low_latency` parameter was used. You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

19 August 2022

Important: Deprecation date for most previous-generation models is now 3 March 2023

Superseded: This deprecation notice is superseded by the 15 February 2023 service update. The end of service date for *all* previous-generation models is now **31 July 2023**.

On 15 March 2022, the previous-generation models for all languages other than Arabic and Japanese were deprecated. At that time, the deprecated models were to remain available until 15 September 2022. To allow users more time to migrate to the appropriate next-generation models, the deprecated models will now remain available until 3 March 2023. As with the initial deprecation notice, the Arabic and Japanese previous-generation models are *not* deprecated. For a complete list of all deprecated models, see the 15 March 2022 service update.

On 3 March 2023, the deprecated models will be removed from the service and the documentation. If you use any of the deprecated models, you must migrate to the equivalent next-generation model by the 3 March 2023.

- For more information about the next-generation models to which you can migrate from each of the deprecated models, see Previous-generation languages and models
- For more information about the next-generation models, see Next-generation languages and models
- For more information about migrating from previous-generation to next-generation models, see Migrating to next-generation models.

Note: When the previous-generation `en-US_BroadbandModel` is removed from service, the next-generation `en-US_Multimedia` model will become the default model for speech recognition requests.

15 August 2022

New French Canadian next-generation multimedia model

The service now offers a next-generation multimedia model for French Canadian: `fr-CA_Multimedia`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Updates to English next-generation telephony models

The English next-generation telephony models have been updated for improved speech recognition:

- `en-AU_Telephony`
- `en-GB_Telephony`
- `en-IN_Telephony`
- `en-US_Telephony`

All of these models continue to support low latency. You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

Italian next-generation multimedia model now supports low latency

The Italian next-generation multimedia model, `it-IT_Multimedia`, now supports low latency. For more information about next-generation models and low latency, see

- Next-generation languages and models

- Customization support for next-generation models
- Low latency

Important: Maximum hours of audio data being reduced for custom acoustic models

Important: The maximum amount of audio data that you can add to a custom acoustic model is being reduced from 200 hours to 50 hours. This change is being phased into different locations from August to September 2022. For information about the schedule for the limit reduction and what it means for existing custom acoustic models that contain more than 50 hours of audio, see Maximum hours of audio.

3 August 2022

Defect fix: Update speech hesitations and hesitation markers documentation

Defect fix: Documentation for speech hesitations and hesitation markers has been updated. Previous-generation models include hesitation markers in place of speech hesitations in transcription results for most languages; smart formatting removes hesitation markers from US English final transcripts. Next-generation models include the actual speech hesitations in transcription results; smart formatting has no effect on their inclusion in final transcription results.

For more information, see:

- Speech hesitations and hesitation markers
- What results does smart formatting affect?

1 June 2022

Updates to multiple next-generation telephony models

The following next-generation telephony models have been updated for improved speech recognition:

- `en-AU_Telephony`
- `en-GB_Telephony`
- `en-IN_Telephony`
- `en-US_Telephony`
- `ko-KR_Telephony`

You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

25 May 2022

New beta `character_insertion_bias` parameter for next-generation models

All next-generation models now support a new beta parameter, `character_insertion_bias`, which is available with all speech recognition interfaces. By default, the service is optimized for each individual model to balance its recognition of candidate strings of different lengths. The model-specific bias is equivalent to 0.0. Each model's default bias is sufficient for most speech recognition requests.

However, certain use cases might benefit from favoring hypotheses with shorter or longer strings of characters. The parameter accepts values between -1.0 and 1.0 that represent a change from a model's default. Negative values instruct the service to favor shorter strings of characters. Positive values direct the service to favor longer strings of characters. For more information, see Character insertion bias.

19 May 2022

New Italian `it-IT_Multimedia` next-generation model

The service now offers a next-generation multimedia model for Italian: `it-IT_Multimedia`. The new model is generally available. It does not support low latency, but it does support language model customization and grammars. For more information about all available next-generation models, see Next-generation languages and models.

Updated Korean telephony and multimedia next-generation models

The existing Korean next-generation models have been updated:

- The `ko-KR_Telephony` model has been updated for improved low-latency support for speech recognition.
- The `ko-KR_Multimedia` model has been updated for improved speech recognition. The model now also supports low latency.

Both models are generally available, and both support language model customization and grammars. You do not need to upgrade custom language models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

Defect fix: Confidence scores are now reported for all transcription results

Defect fix: Confidence scores are now reported for all transcription results. Previously, when the service returned multiple transcripts for a single speech recognition request, confidence scores might not be returned for all transcripts.

11 April 2022

New Brazilian Portuguese `pt-BR_Multimedia` next-generation model

The service now offers a next-generation multimedia model for Brazilian Portuguese: `pt-BR_Multimedia`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Update to German `de-DE_Multimedia` next-generation model to support low latency

The next-generation German model, `de-DE_Multimedia`, now supports low latency. You do not need to upgrade custom models that are based on the updated German base model. For more information about the next-generation models and low latency, see

- Next-generation languages and models
- Low latency

Support for sounds-like is now documented for custom models based on next-generation models

For custom language models that are based on next-generation models, support is now documented for sounds-like specifications for custom words. Support for sounds-likes has been available since late 2021.

Differences exist between the use of the `sounds_like` field for custom models that are based on next-generation and previous-generation models. For more information about using the `sounds_like` field with custom models that are based on next-generation models, see Working with custom words for next-generation models.

Important: Deprecated `customization_id` parameter removed from the documentation

Important: On 9 October 2018, the `customization_id` parameter of all speech recognition requests was deprecated and replaced by the `language_customization_id` parameter. The `customization_id` parameter has now been removed from the documentation for the speech recognition methods:

- `/v1/recognize` for WebSocket requests
- `POST /v1/recognize` for synchronous HTTP requests (including multipart requests)
- `POST /v1/recognitions` for asynchronous HTTP requests

Note: If you use the Watson SDKs, make sure that you have updated any application code to use the `language_customization_id` parameter instead of the `customization_id` parameter. The `customization_id` parameter will no longer be available from the equivalent methods of the SDKs as of their next major release. For more information about the speech recognition methods, see the API & SDK reference.

17 March 2022

Grammar support for next-generation models is now generally available

Grammar support is now generally available (GA) for next-general models that meet the following conditions:

- The models are generally available.
- The models support language model customization.

For more information, see the following topics:

- For more information about the status of grammar support for next-generation models, see Customization support for next-generation models.
- For more information about grammars, see Grammars.

New German next-generation multimedia model

The service now offers a next-generation multimedia model for German: `de-DE_Multimedia`. The new model is generally available. It does not support low latency. It does support language model customization (generally available) and grammars (beta).

For more information about all available next-generation models and their customization support, see

- Next-generation languages and models
- Customization support for next-generation models

Beta next-generation `en-WW_Medical_Telephony` model now supports low latency

The beta next-generation `en-WW_Medical_Telephony` model now supports low latency. For more information about all next-generation models and low latency, see

- Next-generation languages and models
- Low latency

15 March 2022

Important: Deprecation of most previous-generation models

Superseded: This deprecation notice is superseded by the 15 February 2023 service update. The end of service date for *all* previous-generation models is now **31 July 2023**.

Effective 15 March 2022, previous-generation models for all languages other than Arabic and Japanese are deprecated. The deprecated models remain available until 15 September 2022, when they will be removed from the service and the documentation. The Arabic and Japanese previous-generation models are *not* deprecated.

The following previous-generation models are now deprecated:

- Chinese (Mandarin): `zh-CN_NarrowbandModel` and `zh-CN_BroadbandModel`
- Dutch (Netherlands): `nl-NL_NarrowbandModel` and `nl-NL_BroadbandModel`
- English (Australian): `en-AU_NarrowbandModel` and `en-AU_BroadbandModel`
- English (United Kingdom): `en-GB_NarrowbandModel` and `en-GB_BroadbandModel`
- English (United States): `en-US_NarrowbandModel`, `en-US_BroadbandModel`, and `en-US_ShortForm_NarrowbandModel`
- French (Canadian): `fr-CA_NarrowbandModel` and `fr-CA_BroadbandModel`
- French (France): `fr-FR_NarrowbandModel` and `fr-FR_BroadbandModel`
- German: `de-DE_NarrowbandModel` and `de-DE_BroadbandModel`
- Italian: `it-IT_NarrowbandModel` and `it_IT_BroadbandModel`
- Korean: `ko-KR_NarrowbandModel` and `ko-KR_BroadbandModel`
- Portuguese (Brazilian): `pt-BR_NarrowbandModel` and `pt-BR_BroadbandModel`
- Spanish (Argentinian): `es-AR_NarrowbandModel` and `es-AR_BroadbandModel`
- Spanish (Castilian): `es-ES_NarrowbandModel` and `es-ES_BroadbandModel`
- Spanish (Chilean): `es-CL_NarrowbandModel` and `es-CL_BroadbandModel`

- Spanish (Colombian): `es-CO_NarrowbandModel` and `es-CO_BroadbandModel`
- Spanish (Mexican): `es-MX_NarrowbandModel` and `es-MX_BroadbandModel`
- Spanish (Peruvian): `es-PE_NarrowbandModel` and `es-PE_BroadbandModel`

If you use any of these deprecated models, you must migrate to the equivalent next-generation model by the end of service date.

- For more information about the next-generation models to which you can migrate from each of the deprecated models, see Previous-generation languages and models
- For more information about the next-generation models, see Next-generation languages and models
- For more information about migrating from previous-generation to next-generation models, see Migrating to next-generation models.

Note: When the previous-generation `en-US_BroadbandModel` is removed from service on 15 September, the next-generation `en-US_Multimedia` model will become the default model for speech recognition requests.

Next-generation models now support audio-parsing parameters

All next-generation models now support the following audio-parsing parameters as generally available features:

- `end_of_phrase_silence_time` specifies the duration of the pause interval at which the service splits a transcript into multiple final results. For more information, see End of phrase silence time .
- `split_transcript_at_phrase_end` directs the service to split the transcript into multiple final results based on semantic features of the input. For more information, see Split transcript at phrase end .

Defect fix: Correct speaker labels documentation

Defect fix: Documentation of speaker labels included the following erroneous statement in multiple places: *For next-generation models, speaker labels are not supported for use with interim results or low latency.* Speaker labels are supported for use with interim results and low latency for next-generation models. For more information, see Speaker labels.

28 February 2022

Updates to English and French next-generation multimedia models to support low latency

The following multimedia models have been updated to support low latency:

- Australian English: `en-AU_Multimedia`
- UK English: `en-GB_Multimedia`
- US English: `en-US_Multimedia`
- French: `fr-FR_Multimedia`

You do not need to upgrade custom language models that are built on these base models. For more information about the next-generation models and low latency, see

- Next-generation languages and models
- Low latency

New Castilian Spanish next-generation multimedia model

The service now offers a next-generation multimedia model for Castilian Spanish: `es-ES_Multimedia`. The new model supports low latency and is generally available. It also supports language model customization (generally available) and grammars (beta).

For more information about all available next-generation models and their customization support, see

- Next-generation languages and models
- Customization support for next-generation models

11 February 2022

Defect fix: Correct custom model upgrade and base model version documentation

Defect fix: The documentation that describes the upgrade of custom models and the version strings that are used for different versions of base models has been updated. The documentation now states that upgrade for language model customization also applies to next-generation models. Also, the version strings that represent different versions of base models have been updated. And the `base_model_version` parameter can also be used with upgraded next-generation models.

For more information about custom model upgrade, when upgrade is necessary, and how to use older versions of custom models, see

- Upgrading custom models
- Using upgraded custom models for speech recognition

Defect fix: Update capitalization documentation

Defect fix: The documentation that describes the service's automatic capitalization of transcripts has been updated. The service capitalizes appropriate nouns only for the following languages and models:

- All previous-generation US English models
- The next-generation German model

For more information, see Capitalization.

2 February 2022

New beta `en-WW_Medical_Telephony` model is now available

A new beta next-generation `en-WW_Medical_Telephony` is now available. The new model understands terms from the medical and pharmacological domains. Use the model in situations where you need to transcribe common medical terminology such as medicine names, product brands, medical procedures, illnesses, types of doctor, or COVID-19-related terminology. Common use cases include conversations between a patient and a medical provider (for example, a doctor, nurse, or pharmacist).

The new model is available for all supported English dialects: Australian, Indian, UK, and US. The new model supports language model customization and grammars as beta functionality. It supports most of the same parameters as the `en-US_Telephony` model, including `smart_formatting` for US English audio. It does not support the following parameters: `low_latency`, `profanity_filter`, `redaction`, and `speaker_labels`.

For more information, see The English medical telephony model.

Update to Chinese `zh-CN_Telephony` model

The next-generation Chinese model `zh-CN_Telephony` has been updated for improved speech recognition. The model continues to support low latency. By default, the service automatically uses the updated model for all speech recognition requests. For more information about all available next-generation models, see Next-generation languages and models.

If you have custom language models that are based on the updated model, you must upgrade your existing custom models to take advantage of the updates by using the `POST /v1/customizations/{customization_id}/upgrade_model` method. For more information, see Upgrading custom models.

Update to Japanese `ja-JP_Multimedia` next-generation model to support low latency

The next-generation Japanese model `ja-JP_Multimedia` now supports low latency. You can use the `low_latency` parameter with speech recognition requests that use the model. You do not need to upgrade custom models that are based on the updated Japanese base model. For more information about the next-generation models and low latency, see

- Next-generation languages and models
- Low latency

3 December 2021

New Latin American Spanish next-generation telephony model

The service now offers a next-generation telephony model for Latin American Spanish: `es-LA_Telephony`. The new model supports low latency and is generally available.

The `es-LA_Telephony` model applies to all Latin American dialects. It is the equivalent of the previous-generation models that are available for the Argentinian, Chilean, Colombian, Mexican, and Peruvian dialects. If you used a previous-generation model for any of these specific dialects, use the `es-LA_Telephony` model to migrate to the equivalent next-generation model.

For more information about all available next-generation models, see Next-generation languages and models.

Important: Custom language models based on certain next-generation models must be re-created

Important: If you created custom language models based on certain next-generation models, you must re-create the custom models. Until you re-create the custom language models, speech recognition requests that attempt to use the custom models fail with HTTP error code 400.

You need to re-create custom language models that you created based on the following versions of next-generation models:

- For the `en-AU_Telephony` model, custom models that you created from `en-AU_Telephony.v2021-03-03` to `en-AU_Telephony.v2021-10-04`.
- For the `en-GB_Telephony` model, custom models that you created from `en-GB_Telephony.v2021-03-03` to `en-GB_Telephony.v2021-10-04`.
- For the `en-US_Telephony` model, custom models that you created from `en-US_Telephony.v2021-06-17` to `en-US_Telephony.v2021-10-04`.
- For the `en-US_Multimedia` model, custom models that you created from `en-US_Multimedia.v2021-03-03` to `en-US_Multimedia.v2021-10-04`.

To identify the version of a model on which a custom language model is based, use the `GET /v1/customizations` method to list all of your custom language models or the `GET /v1/customizations/{customization_id}` method to list a specific custom language model. The `versions` field of the output shows the base model for a custom language model. For more information, see Listing custom language models.

To re-create a custom language model, first create a new custom model. Then add all of the previous custom model's corpora and custom words to the new model. You can then delete the previous custom model. For more information, see Creating a custom language model.

28 October 2021

New Chinese next-generation telephony model

The service now offers a next-generation telephony model for Mandarin Chinese: `zh-CN_Telephony`. The new model supports low latency and is generally available. For more information about all available next-generation models, see Next-generation languages and models.

New Australian English and UK English next-generation multimedia models

The service now offers the following next-generation multimedia models. The new models are generally available, and neither model supports low latency.

- Australian English: `en-AU_Multimedia`
- UK English: `en-GB_Multimedia`

For more information about all available next-generation models, see Next-generation languages and models.

Updates to multiple next-generation models for improved speech recognition

The following next-generation models have been updated for improved speech recognition:

- Australian English telephony model (`en-AU_Telephony`)
- UK English telephony model (`en-GB_Telephony`)
- US English multimedia model (`en-US_Multimedia`)
- US English telephony model (`en-US_Telephony`)
- Castilian Spanish telephony model (`es-ES_Telephony`)

For more information about all available next-generation models, see Next-generation languages and models.

Grammar support for previous-generation models is now generally available

Grammar support is now generally available (GA) for previous-general models that meet the following conditions:

- The models are generally available.
- The models support language model customization.

For more information, see the following topics:

- For more information about the status of grammar support for previous-generation models, see [Customization support for previous-generation models](#).
- For more information about grammars, see [Grammars](#).

New beta grammar support for next-generation models

Grammar support is now available as beta functionality for all next-generation models. All next-generation models are generally available (GA) and support language model customization. For more information, see the following topics:

- For more information about the status of grammar support for next-generation models, see [Customization support for next-generation models](#).
- For more information about grammars, see [Grammars](#).

Note: Beta support for grammars by next-generation models is available for the Speech to Text service on IBM Cloud only. Grammars are not yet supported for next-generation models on IBM Cloud Pak for Data.

New `custom_acoustic_model` field for supported features

The `GET /v1/models` and `GET /v1/models/{model_id}` methods now report whether a model supports acoustic model customization. The `SupportedFeatures` object now includes an additional field, `custom_acoustic_model`, a boolean that is `true` for a model that supports acoustic model customization and `false` otherwise. Currently, the field is `true` for all previous-generation models and `false` for all next-generation models.

- For more information about these methods, see [Listing information about models](#).
- For more information about support for acoustic model customization, see [Language support for customization](#).

22 October 2021

Defect fix: Address asynchronous HTTP failures

Defect fix: The asynchronous HTTP interface failed to transcribe some audio. In addition, the callback for the request returned status `recognitions.completed_with_results` instead of `recognitions.failed`. This error has been resolved.

6 October 2021

Updates to Czech and Dutch next-generation models

The following next-generation language models have changed as indicated:

- The Czech telephony model, `cs-CZ_Telephony`, is now generally available (GA). The model continues to support low latency.
- The Belgian Dutch telephony model, `nl-BE_Telephony`, has been updated for improved speech recognition. The model continues to support low latency.
- The Netherlands Dutch telephony model, `nl-NL_Telephony`, is now GA. In addition, the model now supports low latency.

For more information about all available next-generation language models, see [Next-generation languages and models](#).

New US HIPAA support for Premium plans in Dallas location

US Health Insurance Portability and Accountability Act (HIPAA) support is now available for Premium plans that are hosted in the Dallas (`us-south`) location. For more information, see [Health Insurance Portability and Accountability Act (HIPAA)](#).

16 September 2021

New beta Czech and Netherlands Dutch next-generation models

The service now supports the following new next-generation language models. Both new models are beta functionality.

- Czech: `cs-CZ_Telephony`. The new model supports low latency.
- Netherlands Dutch: `nl-NL_Telephony`. The new model does not support low latency.

For more information about all available next-generation language models, see Next-generation languages and models.

Updates to Korean and Brazilian Portuguese next-generation models

The following next-generation models have been updated:

- The Korean model `ko-KR_Telephony` now supports low latency.
- The Brazilian Portuguese model `pt-BR_Telephony` has been updated for improved speech recognition.

Defect fix: Correct interim results and low-latency documentation

Defect fix: Documentation that describes the interim results and low-latency features with next-generation models has been rewritten for clarity and correctness. For more information, see the following topics:

- Interim results and low latency, especially Requesting interim results and low latency
- How the service sends recognition results

Defect fix: Improve speakers labels results

Defect fix: When you use speakers labels with next-generation models, the service now identifies the speaker for all words of the input audio, including very short words that have the same start and end timestamps.

31 August 2021

All next-generation models are now generally available

All existing next-generation language models are now generally available (GA). They are supported for use in production environments and applications.

- For more information about all available next-generation language models, see Next-generation languages and models.
- For more information about the features that are supported for each next-generation model, see Supported features for next-generation models.

Language model customization for next-generation models is now generally available

Language model customization is now generally available (GA) for all available next-generation languages and models. Language model customization for next-generation models is supported for use in production environments and applications.

You use the same commands to create, manage, and use custom language models, corpora, and custom words for next-generation models as you do for previous-generation models. But customization for next-generation models works differently from customization for previous-generation models. For custom models that are based on next-generation models:

- The custom models have no concept of out-of-vocabulary (OOV) words.
- Words from corpora are not added to the words resource.
- You cannot currently use the sounds-like feature for custom words.
- You do not need to upgrade custom models when base language models are updated.
- Grammars are not currently supported.

For more information about using language model customization for next-generation models, see

- Understanding customization
- Language support for customization
- Creating a custom language model
- Using a custom language model for speech recognition
- Working with corpora and custom words for next-generation models

Additional topics describe managing custom language models, corpora, and custom words. These operations are the same for custom models based on previous- and next-generation models.

16 August 2021

New beta Indian English, Indian Hindi, Japanese, and Korean next-generation models

The service now supports the following new next-generation language models. All of the new models are beta functionality.

- Indian English: `en-IN_Telephony`. The model supports low latency.
- Indian Hindi: `hi-IN_Telephony`. The model supports low latency.
- Japanese: `ja-JP_Multimedia`. The model does not support low latency.
- Korean: `ko-KR_Multimedia` and `ko-KR_Telephony`. The models do not support low latency.

For more information about the next-generation models and low latency, see Next-generation languages and models and Low latency.

16 July 2021

New beta French next-generation model

The French next-generation language model `fr-FR_Multimedia` is now available. The new model does not support low latency. The model is beta functionality.

Updates to beta US English next-generation model for improved speech recognition

The next-generation US English `en-US_Telephony` model has been updated for improved speech recognition. The updated model continues to be beta functionality.

Defect fix: Update documentation for hesitation markers

Defect fix: The documentation failed to state that next-generation models do not produce hesitation markers. The documentation has been updated to note that only previous-generation models produce hesitation markers. Next-generation models include the actual hesitations in transcription results. For more information, see Speech hesitations and hesitation markers.

15 June 2021

New beta Belgian Dutch next-generation model

The Belgian Dutch (Flemish) next-generation language model `nl-BE_Telephony` is now available. The new model supports low latency. The model is beta functionality. For more information about the next-generation models and about low latency, see Next-generation languages and models and Low latency.

New beta low-latency support for Arabic, Canadian French, and Italian next-generation models

The following existing beta next-generation language models now support low latency:

- Arabic `ar-MS_Telephony` model
- Canadian French `fr-CA_Telephony` model
- Italian `it-IT_Telephony` model

For more information about the next-generation models and about low latency, see Next-generation languages and models and Low latency.

Updates to beta Arabic and Brazilian Portuguese next-generation models for improved speech recognition

The following existing beta next-generation language models have been updated for improved speech recognition:

- Arabic `ar-MS_Telephony` model
- Brazilian Portuguese `pt-BR_Telephony` model

For more information about the next-generation models and about low latency, see Next-generation languages and models and Low latency.

26 May 2021

New beta support for `audio_metrics` parameter for next-generation models

The `audio_metrics` parameter is now supported as beta functionality for use with all next-generation languages and models. For more information, see Audio metrics.

New beta support for `word_confidence` parameter for next-generation models

The `word_confidence` parameter is now supported as beta functionality for use with all next-generation languages and models. For more information, see Word confidence.

Defect fix: Update documentation for next-generation models

Defect fix: The documentation has been updated to correct the following information:

- When you use a next-generation model for speech recognition, final transcription results now include the `confidence` field. The field was always included in final transcription results when you use a previous-generation model. This fix addresses a limitation that was reported for the 12 April 2021 release of the next-generation models.
- The documentation incorrectly stated that using the `smart_formatting` parameter causes the service to remove hesitation markers from final transcription results for Japanese. Smart formatting does not remove hesitation markers from final results for Japanese, only for US English. For more information, see What results does smart formatting affect?

27 April 2021

New beta Arabic and Brazilian Portuguese next-generation models

The service supports two new beta next-generation models:

- The Brazilian Portuguese `pt-BR_Telephony` model, which supports low latency.
- The Arabic (Modern Standard) `ar-MS_Telephony` model, which does not support low latency.

For more information, see Next-generation languages and models.

Updates to beta Castilian Spanish next-generation model for improved speech recognition

The beta next-generation Castilian Spanish `es-ES_Telephony` model now supports the `low_latency` parameter. For more information, see Low latency.

New beta support for speaker labels with next-generation models

The `speaker_labels` parameter is now supported as beta functionality for use with the following next-generation models:

- Australian English `en-AU_Telephony` model
- UK English `en-GB_Telephony` model
- US English `en-US_Multimedia` and `en-US_Telephony` models
- German `de-DE_Telephony` model
- Castilian Spanish `es-ES_Telephony` model

With the next generation models, the `speaker_labels` parameter is not supported for use with the `interim_results` or `low_latency` parameters at this time. For more information, see Speaker labels.

New HTTP error code for use of `word_confidence` with next-generation models

The `word_confidence` parameter is not supported for use with next-generation models. The service now returns the following 400 error code if you use the `word_confidence` parameter with a next-generation model for speech recognition:

```
{
  "error": "word_confidence is not a supported feature for model {model}",
  "code": 400,
  "code_description": "Bad Request"
}
```

12 April 2021

New beta next-generation language models and `low_latency` parameter

The service now supports a growing number of next-generation language models. The next-generation *multimedia* and *telephony* models improve upon the speech recognition capabilities of the service's previous generation of broadband and narrowband models. The new models leverage deep neural networks and bidirectional analysis to achieve both higher throughput and greater transcription accuracy. At this time, the next-generation models support only a limited number of languages and speech recognition features. The supported languages, models, and features will increase with future releases. The next-generation models are beta functionality.

Many of the next-generation models also support a new `low_latency` parameter that lets you request faster results at the possible expense of reduced transcription quality. When low latency is enabled, the service curtails its analysis of the audio, which can reduce the accuracy of the transcription. This trade-off might be acceptable if your application requires lower response time more than it does the highest possible accuracy.The `low_latency` parameter is beta functionality.

The `low_latency` parameter impacts your use of the `interim_results` parameter with the WebSocket interface. Interim results are available only for those next-generation models that support low latency, and only if both the `interim_results` and `low_latency` parameters are set to `true`.

- For more information about the next-generation models and their capabilities, see Next-generation languages and models.
- For more information about language support for next-generation models and about which next-generation models support low latency, see Supported next-generation language models.
- For more information about feature support for next-generation models, see Supported features for next-generation models.
- For more information about the `low_latency` parameter, see Low latency.
- For more information about the interaction between the `low_latency` and `interim_results` parameters for next-generation models, see Requesting interim results and low latency.

17 March 2021

Defect fix: Fix limitation for asynchronous HTTP interface

Defect fix: The limitation that was reported with the asynchronous HTTP interface in the Dallas (`us-south`) location on 16 December 2020 has been addressed. Previously, a small percentage of jobs were entering infinite loops that prevented their execution. Asynchronous HTTP requests in the Dallas data center no longer experience this limitation.

2 December 2020

Arabic model renamed to `ar-MS_BroadbandModel`

The Arabic language broadband model is now named `ar-MS_BroadbandModel`. The former name, `ar-AR_BroadbandModel`, is deprecated. It will continue to function for at least one year but might be removed at a future date. You are encouraged to migrate to the new name at your earliest convenience.

2 November 2020

Canadian French models now generally available

The Canadian French models, `fr-CA_BroadbandModel` and `fr-CA_NarrowbandModel`, are now generally available (GA). They were previously beta. They also now support language model and acoustic model customization.

- For more information about supported languages and models, see Previous-generation languages and models.
- For more information about language support for customization, see Language support for customization.

22 October 2020

Australian English models now generally available

The Australian English models, `en-AU_BroadbandModel` and `en-AU_NarrowbandModel`, are now generally available (GA). They were previously beta. They also now support language model and acoustic model customization.

- For more information about supported languages and models, see Previous-generation languages and models.
- For more information about language support for customization, see Language support for customization.

Updates to Brazilian Portuguese models for improved speech recognition

The Brazilian Portuguese models, `pt-BR_BroadbandModel` and `pt-BR_NarrowbandModel`, have been updated for improved speech recognition. By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on the models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

The `split_transcript_at_phrase_end` parameter now generally available for all languages

The speech recognition parameter `split_transcript_at_phrase_end` is now generally available (GA) for all languages. Previously, it was generally available only for US and UK English. For more information, see Split transcript at phrase end.

7 October 2020

Updates to Japanese broadband model for improved speech recognition

The `ja-JP_BroadbandModel` model has been updated for improved speech recognition. By default, the service automatically uses the updated model for all speech recognition requests. If you have custom language or custom acoustic models that are based on this model, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

30 September 2020

Updates to pricing plans for the service

The pricing plans for the service have changed:

- The service continues to offer a Lite plan that provides basic no-charge access to limited minutes of speech recognition per month.
- The service offers a new Plus plan that provides a simple tiered pricing model and access to the service's customization capabilities.
- The service offers a new Premium plan that provides significantly greater capacity and enhanced features.

The Plus plan replaces the Standard plan. The Standard plan continues to be available for purchase for a short time. It also continues to be available indefinitely to existing users of the plan with no change in their pricing. Existing users can upgrade to the Plus plan at any time.

For more information about the available pricing plans, see the following resources:

- For general information about the pricing plans and answers to common questions, see the Pricing FAQs.
- For more information about the pricing plans or to purchase a plan, see the Speech to Text service in the IBM Cloud® Catalog.

20 August 2020

New Canadian French models

The service now offers beta broadband and narrowband models for Canadian French:

- `fr-CA_BroadbandModel`
- `fr-CA_NarrowbandModel`

The new models do not support language model or acoustic model customization, speaker labels, or smart formatting. For more information about these and all supported models, see Supported previous-generation language models .

5 August 2020

New Australian English models

The service now offers beta broadband and narrowband models for Australian English:

- `en-AU_BroadbandModel`
- `en-AU_NarrowbandModel`

The new models do not support language model or acoustic model customization, or smart formatting. The new models do support speakers labels. For more information, see

- Supported previous-generation language models
- Speaker labels

Updates to multiple models for improved speech recognition

The following models have been updated for improved speech recognition:

- French broadband model (`fr-FR_BroadbandModel`)
- German broadband (`de-DE_BroadbandModel`) and narrowband (`de-DE_NarrowbandModel`) models
- UK English broadband (`en-GB_BroadbandModel`) and narrowband (`en-GB_NarrowbandModel`) models
- US English short-form narrowband (`en-US_ShortForm_NarrowbandModel`) model

By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on these models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

Hesitation marker for German changed

The hesitation marker that is used for the updated German broadband and narrowband models has changed from `[hesitation]` to `%HESITATION`. For more information, see Speech hesitations and hesitation markers .

4 June 2020

Defect fix: Improve latency for custom language models with many grammars

Defect fix: The latency issue for custom language models that contain a large number of grammars has been resolved. When initially used for speech recognition, such custom models could take multiple seconds to load. The custom models now load much faster, greatly reducing latency when they are used for recognition.

28 April 2020

Updates to Italian models for improved speech recognition

The Italian broadband (`it-IT_BroadbandModel`) and narrowband (`it-IT_NarrowbandModel`) models have been updated for improved speech recognition. By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on these models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

Dutch and Italian models now generally available

The Dutch and Italian language models are now generally available (GA) for speech recognition and for language model and acoustic model customization:

- Dutch broadband model (`nl-NL_BroadbandModel`)
- Dutch narrowband model (`nl-NL_NarrowbandModel`)
- Italian broadband model (`it-IT_BroadbandModel`)
- Italian narrowband model (`it-IT_NarrowbandModel`)

For more information about all available language models, see

- Supported previous-generation language models
- Language support for customization

1 April 2020

Acoustic model customization now generally available

Acoustic model customization is now generally available (GA) for all supported languages. As with custom language models, IBM does not charge for creating or hosting a custom acoustic model. You are charged only for using a custom model with a speech recognition request.

Using a custom language model, a custom acoustic model, or both types of model for transcription incurs an add-on charge of $0.03 (USD) per minute. This charge is in addition to the standard usage charge of $0.02 (USD) per minute, and it applies to all languages supported by the customization interface. So the total charge for using one or more custom models for speech recognition is $0.05 (USD) per minute.

- For more information about support for individual language models, see Language support for customization.
- For more information about pricing, see the pricing page for the Speech to Text service or the Pricing FAQs.

16 March 2020

Speaker labels now supported for German and Korean

The service now supports speaker labels (the `speaker_labels` parameter) for German and Korean language models. Speaker labels identify which individuals spoke which words in a multi-participant exchange. For more information, see Speaker labels.

Activity Tracker now supported for asynchronous HTTP interface

The service now supports the use of Activity Tracker events for all operations of the asynchronous HTTP interface. IBM Cloud Activity Tracker records user-initiated activities that change the state of a service in IBM Cloud®. For more information, see Activity Tracker events.

24 February 2020

Updates to multiple models for improved speech recognition

The following models have been updated for improved speech recognition:

- Dutch broadband model (`nl-NL_BroadbandModel`)
- Dutch narrowband model (`nl-NL_NarrowbandModel`)
- Italian broadband model (`it-IT_BroadbandModel`)
- Italian narrowband model (`it-IT_NarrowbandModel`)
- Japanese narrowband model (`ja-JP_NarrowbandModel`)
- US English broadband model (`en-US_BroadbandModel`)

By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on the models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

Language model customization now available for Dutch and Italian

Language model customization is now supported for Dutch and Italian with the new versions of the following models:

- Dutch broadband model (`nl-NL_BroadbandModel`)
- Dutch narrowband model (`nl-NL_NarrowbandModel`)
- Italian broadband model (`it-IT_BroadbandModel`)
- Italian narrowband model (`it-IT_NarrowbandModel`)

For more information, see

- Parsing of Dutch, English, French, German, Italian, Portuguese, and Spanish
- Guidelines for Dutch, French, German, Italian, Portuguese, and Spanish

Because the Dutch and Italian models are beta, their support for language model customization is also beta.

Japanese narrowband model now includes some multigram word units

The Japanese narrowband model (`ja-JP_NarrowbandModel`) now includes some multigram word units for digits and decimal fractions. The service returns these multigram units regardless of whether you enable smart formatting. The smart formatting feature understands and returns the multigram units that the model generates. If you apply your own post-processing to transcription results, you need to handle these units appropriately. For more information, see Japanese in the smart formatting documentation.

New speech activity detection and background audio suppression parameters for speech recognition

The service now offers two new optional parameters for controlling the level of speech activity detection. The parameters can help ensure that only relevant audio is processed for speech recognition.

- The `speech_detector_sensitivity` parameter adjusts the sensitivity of speech activity detection. You can use the parameter to suppress word insertions from music, coughing, and other non-speech events.
- The `background_audio_suppression` parameter suppresses background audio based on its volume to prevent it from being transcribed or otherwise interfering with speech recognition. You can use the parameter to suppress side conversations or background noise.

You can use the parameters individually or together. They are available for all interfaces and for most language models. For more information about the parameters, their allowable values, and their effect on the quality and latency of speech recognition, see Speech activity detection.

Activity Tracker now supported for customization interfaces

The service now supports the use of Activity Tracker events for all customization operations. IBM Cloud Activity Tracker records user-initiated activities that change the state of a service in IBM Cloud. You can use this service to investigate abnormal activity and critical actions and to comply with regulatory audit requirements. In addition, you can be alerted about actions as they happen. For more information, see Activity Tracker events.

Defect fix: Correct generation of processing metrics with WebSocket interface

Defect fix: The WebSocket interface now works seamlessly when generating processing metrics. Previously, processing metrics could continue to be delivered after the client sent a `stop` message to the service.

18 December 2019

New beta Italian models available

The service now offers beta broadband and narrowband models for the Italian language:

- `it-IT_BroadbandModel`
- `it-IT_NarrowbandModel`

These language models support acoustic model customization. They do not support language model customization. Because they are beta, these models might not be ready for production use and are subject to change. They are initial offerings that are expected to improve in quality with time and usage.

For more information, see the following sections:

- Supported previous-generation language models
- Language support for customization

New `end_of_phrase_silence_time` parameter for speech recognition

For speech recognition, the service now supports the `end_of_phrase_silence_time` parameter. The parameter specifies the duration of the pause interval at which the service splits a transcript into multiple final results. Each final result indicates a pause or extended silence that exceeds the pause interval. For most languages, the default pause interval is 0.8 seconds; for Chinese the default interval is 0.6 seconds.

You can use the parameter to effect a trade-off between how often a final result is produced and the accuracy of the transcription. Increase the interval when accuracy is more important than latency. Decrease the interval when the speaker is expected to say short phrases or single words.

For more information, see End of phrase silence time.

New `split_transcript_at_phrase_end` parameter for speech recognition

For speech recognition, the service now supports the `split_transcript_at_phrase_end` parameter. The parameter directs the service to split the transcript into multiple final results based on semantic features of the input, such as at the conclusion of sentences. The service bases its understanding of semantic features on the base language model that you use with a request. Custom language models and grammars can also influence how and where the service splits a transcript.

The parameter causes the service to add an `end_of_utterance` field to each final result to indicate the motivation for the split: `full_stop`, `silence`, `end_of_data`, or `reset`.

For more information, see Split transcript at phrase end.

12 December 2019

Full support for IBM Cloud IAM

The Speech to Text service now supports the full implementation of IBM Cloud Identity and Access Management (IAM). API keys for IBM Watson® services are no longer limited to a single service instance. You can create access policies and API keys that apply to more than one service, and you can grant access between services. For more information about IAM, see Authenticating to Watson services.

To support this change, the API service endpoints use a different domain and include the service instance ID. The pattern is `api.{location}.speech-to-text.watson.cloud.ibm.com/instances/{instance_id}`.

- Example HTTP URL for an instance hosted in the Dallas location:

  ```
  https://api.us-south.speech-to-text.watson.cloud.ibm.com/instances/6bbda3b3-d572-45e1-8c54-22d6ed9e52c2
  ```

- Example WebSocket URL for an instance hosted in the Dallas location:

  ```
  wss://api.us-south.speech-to-text.watson.cloud.ibm.com/instances/6bbda3b3-d572-45e1-8c54-22d6ed9e52c2
  ```

For more information about the URLs, see the API & SDK reference.

These URLs do not constitute a breaking change. The new URLs work for both your existing service instances and for new instances. The original URLs continue to work on your existing service instances for at least one year, until December 2020.

New network and data security features available

Support for the following new network and data security features is now available:

- *Support for private network endpoints*

  Users of Premium plans can create private network endpoints to connect to the Speech to Text service over a private network. Connections to private network endpoints do not require public internet access. For more information, see [Public and private network endpoints]().

- *Support for data encryption with customer-managed keys*

  Users of new Premium and Dedicated instances can integrate IBM® Key Protect for IBM Cloud® with the Speech to Text service to encrypt your data and manage encryption keys. For more information, see [Protecting sensitive information in your Watson service]().

10 December 2019

New beta Netherlands Dutch models available

The service now offers beta broadband and narrowband models for the Netherlands Dutch language:

- `nl-NL_BroadbandModel`
- `nl-NL_NarrowbandModel`

These language models support acoustic model customization. They do not support language model customization. Because they are beta, these models might not be ready for production use and are subject to change. They are initial offerings that are expected to improve in quality with time and usage.

For more information, see the following sections:

- [Supported previous-generation language models]()
- [Language support for customization]()

25 November 2019

Updates to speaker labels for improved identification of individual speakers

Speaker labels are updated to improve the identification of individual speakers for further analysis of your audio sample. For more information about the speaker labels feature, see [Speaker labels](). For more information about the improvements to the feature, see [IBM Research AI Advances Speaker Diarization in Real Use Cases]().

12 November 2019

New Seoul location now available

The Speech to Text service is now available in the IBM Cloud Seoul location (**kr-seo**). As with other locations, the IBM Cloud location uses token-based IAM authentication. All new services instances that you create in this location use IAM authentication.

1 November 2019

New limits on maximum number of custom models

You can create no more than 1024 custom language models and no more than 1024 custom acoustic models per owning credentials. For more information, see [Maximum number of custom models]().

1 October 2019

New US HIPAA support for Premium plans in Washington, DC, location

US HIPAA support is available for Premium plans that are hosted in the Washington, DC (**us-east**), location and are created on or after 1 April 2019. For more information, see [US Health Insurance Portability and Accountability Act (HIPAA)](#).

22 August 2019

Defect fix: Multiple small improvements

The service was updated for small defect fixes and improvements.

30 July 2019

New models for Spanish dialects now available

The service now offers broadband and narrowband language models in six Spanish dialects:

- Argentinian Spanish (`es-AR_BroadbandModel` and `es-AR_NarrowbandModel`)
- Castilian Spanish (`es-ES_BroadbandModel` and `es-ES_NarrowbandModel`)
- Chilean Spanish (`es-CL_BroadbandModel` and `es-CL_NarrowbandModel`)
- Colombian Spanish (`es-CO_BroadbandModel` and `es-CO_NarrowbandModel`)
- Mexican Spanish (`es-MX_BroadbandModel` and `es-MX_NarrowbandModel`)
- Peruvian Spanish (`es-PE_BroadbandModel` and `es-PE_NarrowbandModel`)

The Castilian Spanish models are not new. They are generally available (GA) for speech recognition and language model customization, and beta for acoustic model customization.

The other five dialects are new and are beta for all uses. Because they are beta, these additional dialects might not be ready for production use and are subject to change. They are initial offerings that are expected to improve in quality with time and usage.

For more information, see the following sections:

- [Supported previous-generation language models](#)
- [Language support for customization](#)

24 June 2019

Updates to Brazilian Portuguese and US English models for improved speech recognition

The following narrowband models have been updated for improved speech recognition:

- Brazilian Portuguese narrowband model (`pt-BR_NarrowbandModel`)
- US English narrowband model (`en-US_NarrowbandModel`)

By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on the models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see [Upgrading custom models](#).

New support for concurrent requests to update different custom acoustic models

The service now allows you to submit multiple simultaneous requests to add different audio resources to a custom acoustic model. Previously, the service allowed only one request at a time to add audio to a custom model.

New `updated` field for methods that list custom models

The output of the HTTP `GET` methods that list information about custom language and custom acoustic models now includes an `updated` field. The field indicates the date and time in Coordinated Universal Time (UTC) at which the custom model was last modified.

Change to schema for warnings associated with custom model training

The schema changed for a warning that is generated by a custom model training request when the `strict` parameter is set to `false`. The names of the fields changed from `warning_id` and `description` to `code` and `message`, respectively. For more information, see the API & SDK reference.

10 June 2019

Processing metrics not available with synchronous HTTP interface

Processing metrics are available only with the WebSocket and asynchronous HTTP interfaces. They are not supported with the synchronous HTTP interface. For more information, see Processing metrics.

17 May 2019

New processing metrics and audio metrics features for speech recognition

The service now offers two types of optional metrics with speech recognition requests:

- Processing metrics provide detailed timing information about the service's analysis of the input audio. The service returns the metrics at specified intervals and with transcription events, such as interim and final results. Use the metrics to gauge the service's progress in transcribing the audio.
- Audio metrics provide detailed information about the signal characteristics of the input audio. The results provide aggregated metrics for the entire input audio at the conclusion of speech processing. Use the metrics to determine the characteristics and quality of the audio.

You can request both types of metrics with any speech recognition request. By default, the service returns no metrics for a request.

Updates to Japanese broadband model for improved speech recognition

The Japanese broadband model (`ja-JP_BroadbandModel`) has been updated for improved speech recognition. By default, the service automatically uses the updated model for all speech recognition requests. If you have custom language or custom acoustic models that are based on the model, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

10 May 2019

Updates to Spanish models for improved speech recognition

The Spanish language models have been updated for improved speech recognition:

- `es-ES_BroadbandModel`
- `es-ES_NarrowbandModel`

By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on the models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see [Upgrading custom models](#).

19 April 2019

New `strict` parameter for custom model training now available

The training methods of the customization interface now include a `strict` query parameter that indicates whether training is to proceed if a custom model contains a mix of valid and invalid resources. By default, training fails if a custom model contains one or more invalid resources. Set the parameter to `false` to allow training to proceed as long as the model contains at least one valid resource. The service excludes invalid resources from the training.

- For more information about using the `strict` parameter with the `POST /v1/customizations/{customization_id}/train` method, see Train the custom language model and Training failures.
- For more information about using the `strict` parameter with the `POST /v1/acoustic_customizations/{customization_id}/train` method, see Train the custom acoustic model and Training failures.

New limits on maximum number of out-of-vocabulary words for custom language models

You can now add a maximum of 90 thousand out-of-vocabulary (OOV) words to the words resource of a custom language model. The previous maximum was 30 thousand OOV words. This figure includes OOV words from all sources (corpora, grammars, and individual custom words that you add directly). You can add a maximum of 10 million total words to a custom model from all sources. For more information, see How much data do I need?.

3 April 2019

New limits on maximum amount of audio for custom acoustic models

Custom acoustic models now accept a maximum of 200 hours of audio. The previous maximum limit was 100 hours of audio.

21 March 2019

Visibility of service credentials now restricted by role

Users can now see only service credential information that is associated with the role that has been assigned to their IBM Cloud account. For example, if you are assigned a `reader` role, any `writer` or higher levels of service credentials are no longer visible.

This change does not affect API access for users or applications with existing service credentials. The change affects only the viewing of credentials within IBM Cloud.

15 March 2019

New support for A-law audio format

The service now supports audio in the A-law (`audio/alaw`) format. For more information, see audio/alaw format.

11 March 2019

Change to passing value of `0` for `max_alternatives` parameter

For the `max_alternatives` parameter, the service again accepts a value of `0`. If you specify `0`. the service automatically uses the default value, `1`. A change made for the March 4 service update caused a value of `0` to return an error. (The service returns an error if you specify a negative value.)

Change to passing value of `0` for `word_alternatives_threshold` parameter

For the `word_alternatives_threshold` parameter, the service again accepts a value of `0`. A change made for the March 4 service update caused a value of `0` to return an error. (The service returns an error if you specify a negative value.)

New limit on maximum precision for confidence scores

The service now returns all confidence scores with a maximum precision of two decimal places. This change includes confidence scores for transcripts, word confidence, word alternatives, keyword results, and speaker labels.

4 March 2019

Updates to Brazilian Portuguese, French, and Spanish narrowband models for improved speech recognition

The following narrowband language models have been updated for improved speech recognition:

- Brazilian Portuguese narrowband model (`pt-BR_NarrowbandModel`)
- French French model (`fr-FR_NarrowbandModel`)
- Spanish narrowband model (`es-ES_NarrowbandModel`)

By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on the models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

28 January 2019

New support for IBM Cloud IAM by WebSocket interface

The WebSocket interface now supports token-based Identity and Access Management (IAM) authentication from browser-based JavaScript code. The limitation to the contrary has been removed. To establish an authenticated connection with the WebSocket `/v1/recognize` method:

- If you use IAM authentication, include the `access_token` query parameter.
- If you use Cloud Foundry service credentials, include the `watson-token` query parameter.

For more information, see Open a connection.

20 December 2018

New beta grammars feature for custom language models now available

The service now supports grammars for speech recognition. Grammars are available as beta functionality for all languages that support language model customization. You can add grammars to a custom language model and use them to restrict the set of phrases that the service can recognize from audio. You can define a grammar in Augmented Backus-Naur Form (ABNF) or XML Form.

The following four methods are available for working with grammars:

- `POST /v1/customizations/{customization_id}/grammars/{grammar_name}` adds a grammar file to a custom language model.
- `GET /v1/customizations/{customization_id}/grammars` lists information about all grammars for a custom model.
- `GET /v1/customizations/{customization_id}/grammars/{grammar_name}` returns information about a specified grammar for a custom model.
- `DELETE /v1/customizations/{customization_id}/grammars/{grammar_name}` removes an existing grammar from a custom model.

You can use a grammar for speech recognition with the WebSocket and HTTP interfaces. Use the `language_customization_id` and `grammar_name` parameters to identify the custom model and the grammar that you want to use. Currently, you can use only a single grammar with a speech recognition request.

For more information about grammars, see the following documentation:

- Using grammars with custom language models
- Understanding grammars
- Adding a grammar to a custom language model
- Using a grammar for speech recognition
- Managing grammars
- Example grammars

For information about all methods of the interface, see the API & SDK reference.

New numeric redaction feature for US English, Japanese, and Korean now available

A new numeric redaction feature is now available to mask numbers that have three or more consecutive digits. Redaction is intended to remove sensitive personal information, such as credit card numbers, from transcripts. You enable the feature by setting the `redaction` parameter to `true` on a recognition request. The feature is beta functionality that is available for US English, Japanese, and Korean only. For more information, see Numeric redaction.

New French and German narrowband models now available

The following new German and French language models are now available with the service:

- French narrowband model (`fr-FR_NarrowbandModel`)
- German narrowband model (`de-DE_NarrowbandModel`)

Both new models support language model customization (GA) and acoustic model customization (beta). For more information, see Language support for customization.

New US English `en-US_ShortForm_NarrowbandModel` now available

A new US English language model, `en-US_ShortForm_NarrowbandModel`, is now available. The new model is intended for use in Interactive Voice Response and Automated Customer Support solutions. The model supports language model customization (GA) and acoustic model customization (beta). For more information, see The US English short-form model.

Updates to UK English and Spanish narrowband models for improved speech recognition

The following language models have been updated for improved speech recognition:

- UK English narrowband model (`en-GB_NarrowbandModel`)
- Spanish narrowband model (`es-ES_NarrowbandModel`)

By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on the models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

New support for G.279 audio format

The service now supports audio in the G.729 (`audio/g729`) format. The service supports only G.729 Annex D for narrowband audio. For more information, see audio/g729 format.

Speakers labels feature now available for UK English narrowband model

The speaker labels feature is now available for the narrowband model for UK English (`en-GB_NarrowbandModel`). The feature is beta functionality for all supported languages. For more information, see Speaker labels.

New limits on maximum amount of audio for custom acoustic models

The maximum amount of audio that you can add to a custom acoustic model has increased from 50 hours to 100 hours.

13 December 2018

New London location now available

The Speech to Text service is now available in the IBM Cloud London location (**eu-gb**). Like all locations, London uses token-based IAM authentication. All new services instances that you create in this location use IAM authentication.

12 November 2018

New support for smart formatting for Japanese speech recognition

The service now supports smart formatting for Japanese speech recognition. Previously, the service supported smart formatting for US English and Spanish only. The feature is beta functionality for all supported languages. For more information, see Smart formatting.

7 November 2018

New Tokyo location now available

The Speech to Text service is now available in the IBM Cloud Tokyo location (**jp-tok**). Like all locations, Tokyo uses token-based IAM authentication. All new services instances that you create in this location use IAM authentication.

30 October 2018

New support for token-based IBM Cloud IAM

The Speech to Text service has migrated to token-based IAM authentication for all locations. All IBM Cloud services now use IAM authentication. The Speech to Text service migrated in each location on the following dates:

- Dallas (**us-south**): October 30, 2018
- Frankfurt (**eu-de**): October 30, 2018
- Washington, DC (**us-east**): June 12, 2018
- Sydney (**au-syd**): May 15, 2018

The migration to IAM authentication affects new and existing service instances differently:

- *All new service instances that you create in any location* now use IAM authentication to access the service. You can pass either a bearer token or an API key: Tokens support authenticated requests without embedding service credentials in every call; API keys use HTTP basic authentication. When you use any of the Watson SDKs, you can pass the API key and let the SDK manage the lifecycle of the tokens.
- *Existing service instances that you created in a location before the indicated migration date* continue to use the `{username}` and `{password}` from their previous Cloud Foundry service credentials for authentication until you migrate them to use IAM authentication. For more information about migrating to IAM authentication, see Migrating Watson services from Cloud Foundry .

For more information, see the following documentation:

- To learn which authentication mechanism your service instance uses, view your service credentials by clicking the instance on the IBM Cloud dashboard.
- For more information about using IAM tokens with Watson services, see Authenticating to Watson services.
- For examples that use IAM authentication, see the API & SDK reference .

9 October 2018

Important updates to pricing charges for speech recognition requests

As of 1 October 2018, you are now charged for all audio that you pass to the service for speech recognition. The first one thousand minutes of audio that you send each month are no longer free. For more information about the pricing plans for the service, see the Speech to Text service in the IBM Cloud Catalog.

The `Content-Type` header now optional for most speech recognition requests

The `Content-Type` header is now optional for most speech recognition requests. The service now automatically detects the audio format (MIME type) of most audio. You must continue to specify the content type for the following formats:

- `audio/basic`
- `audio/l16`
- `audio/mulaw`

Where indicated, the content type that you specify for these formats must include the sampling rate and can optionally include the number of channels and the endianness of the audio. For all other audio formats, you can omit the content type or specify a content type of `application/octet-stream` to have the service auto-detect the format.

When you use the `curl` command to make a speech recognition request with the HTTP interface, you must specify the audio format with the `Content-Type` header, specify `"Content-Type: application/octet-stream"`, or specify `"Content-Type:"`. If you omit the header entirely, `curl` uses a default value of `application/x-www-form-urlencoded`. Most of the examples in this documentation continue to specify the format for speech recognition requests regardless of whether it's required.

This change applies to the following methods:

- `/v1/recognize` for WebSocket requests. The `content-type` field of the text message that you send to initiate a request over an open WebSocket connection is now optional.
- `POST /v1/recognize` for synchronous HTTP requests. The `Content-Type` header is now optional. (For multipart requests, the `part_content_type` field of the JSON metadata is also now optional.)
- `POST /v1/recognitions` for asynchronous HTTP requests. The `Content-Type` header is now optional.

For more information, see Audio formats.

Updates to Brazilian Portuguese broadband model for improved speech recognition

The Brazilian Portuguese broadband model, `pt-BR_BroadbandModel`, was updated for improved speech recognition. By default, the service automatically uses the updated model for all recognition requests. If you have custom language or custom acoustic models that are based on this model, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

The `customization_id` parameter renamed to `language_customization_id`

The `customization_id` parameter of the speech recognition methods is deprecated and will be removed in a future release. To specify a custom language model for a speech recognition request, use the `language_customization_id` parameter instead. This change applies to the following methods:

- `/v1/recognize` for WebSocket requests
- `POST /v1/recognize` for synchronous HTTP requests (including multipart requests)
- `POST /v1/recognitions` for asynchronous HTTP requests

10 September 2018

New German broadband model

The service now supports a German broadband model, `de-DE_BroadbandModel`. The new German model supports language model customization (generally available) and acoustic model customization (beta).

- For information about how the service parses corpora for German, see Parsing of Dutch, English, French, German, Italian, Portuguese, and Spanish.
- For more information about creating sounds-like pronunciations for custom words in German, see Guidelines for Dutch, French, German,

Italian, Portuguese, and Spanish.

Language model customization now available for Brazilian Portuguese

The existing Brazilian Portuguese models, `pt-BR_BroadbandModel` and `pt-BR_NarrowbandModel`, now support language model customization (generally available). The models were not updated to enable this support, so no upgrade of existing custom acoustic models is required.

- For information about how the service parses corpora for Brazilian Portuguese, see Parsing of Dutch, English, French, German, Italian, Portuguese, and Spanish.
- For more information about creating sounds-like pronunciations for custom words in Brazilian Portuguese, see Guidelines for Dutch, French, German, Italian, Portuguese, and Spanish.

Updates to US English and Japanese models for improved speech recognition

New versions of the US English and Japanese broadband and narrowband models are available:

- US English broadband model (`en-US_BroadbandModel`)
- US English narrowband model (`en-US_NarrowbandModel`)
- Japanese broadband model (`ja-JP_BroadbandModel`)
- Japanese narrowband model (`ja-JP_NarrowbandModel`)

The new models offer improved speech recognition. By default, the service automatically uses the updated models for all recognition requests. If you have custom language or custom acoustic models that are based on these models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

Keyword spotting and word alternatives features now generally available

The keyword spotting and word alternatives features are now generally available (GA) rather than beta functionality for all languages. For more information, see

- Keyword spotting
- Word alternatives

Defect fix: Improve documentation for customization interface

Defect fix: The following known issues that were associated with the customization interface have been resolved and are fixed in production. The following information is preserved for users who may have encountered the problems in the past.

- If you add data to a custom language or custom acoustic model, you must retrain the model before using it for speech recognition. The problem shows up in the following scenario:

  1. The user creates a new custom model (language or acoustic) and trains the model.

  2. The user adds additional resources (words, corpora, or audio) to the custom model but does not retrain the model.

  3. The user cannot use the custom model for speech recognition. The service returns an error of the following form when used with a speech recognition request:

     ```
     {
       "code_description": "Bad Request",
       "code": 400,
       "error": "Requested custom language model is not available.
                 Please make sure the custom model is trained."
     }
     ```

  To work around this issue, the user must retrain the custom model on its latest data. The user can then use the custom model with speech recognition.

- Before training an existing custom language or custom acoustic model, you must upgrade it to the latest version of its base model. The problem shows up in the following scenario:

  1. The user has an existing custom model (language or acoustic) that is based on a model that has been updated.
  2. The user trains the existing custom model against the old version of the base model without first upgrading to the latest version of the base model.

3. The user cannot use the custom model for speech recognition.

To work around this issue, the user must use the `POST /v1/customizations/{customization_id}/upgrade_model` or `POST /v1/acoustic_customizations/{customization_id}/upgrade_model` method to upgrade the custom model to the latest version of its base model. The user can then use the custom model with speech recognition.

7 September 2018

Session-based interface no longer available

The session-based HTTP REST interface is no longer supported. All information related to sessions is removed from the documentation. The following methods are no longer available:

- `POST /v1/sessions`
- `POST /v1/sessions/{session_id}/recognize`
- `GET /v1/sessions/{session_id}/recognize`
- `GET /v1/sessions/{session_id}/observe_result`
- `DELETE /v1/sessions/{session_id}`

If your application uses the sessions interface, you must migrate to one of the remaining HTTP REST interfaces or to the WebSocket interface. For more information, see the service update for 8 August 2018.

8 August 2018

Deprecation notice for session-based speech recognition interface

The session-based HTTP REST interface is deprecated as of **August 8, 2018**. All methods of the sessions API will be removed from service as of **September 7, 2018**, after which you will no longer be able to use the session-based interface. This notice of immediate deprecation and 30-day removal applies to the following methods:

- `POST /v1/sessions`
- `POST /v1/sessions/{session_id}/recognize`
- `GET /v1/sessions/{session_id}/recognize`
- `GET /v1/sessions/{session_id}/observe_result`
- `DELETE /v1/sessions/{session_id}`

If your application uses the sessions interface, you must migrate to one of the following interfaces by September 7:

- For stream-based speech recognition (including live-use cases), use the WebSocket interface, which provides access to interim results and the lowest latency.
- For file-based speech recognition, use one of the following interfaces:
  - For shorter files of up to a few minutes of audio, use either the synchronous HTTP interface `(POST /v1/recognize`) or the asynchronous HTTP interface (`POST /v1/recognitions`).
  - For longer files of more than a few minutes of audio, use the asynchronous HTTP interface. The asynchronous HTTP interface accepts as much as 1 GB of audio data with a single request.

The WebSocket and HTTP interfaces provide the same results as the sessions interface (only the WebSocket interface provides interim results). You can also use one of the Watson SDKs, which simplify application development with any of the interfaces. For more information, see the API & SDK reference.

13 July 2018

Updates to Spanish narrowband model for improved speech recognition

The Spanish narrowband model, `es-ES_NarrowbandModel`, was updated for improved speech recognition. By default, the service automatically uses the updated model for all recognition requests. If you have custom language or custom acoustic models that are based on this model, you must upgrade your custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

As of this update, the following two versions of the Spanish narrowband model are available:

- `es_ES.8kHz.general.lm20180522235959.am20180522235959` (current)
- `es_ES.8kHz.general.lm20180308235959.am20180308235959` (previous)

The following version of the model is no longer available:

- `es_ES.8kHz.general.lm20171031235959.am20171031235959`

A recognition request that attempts to use a custom model that is based on the now unavailable base model uses the latest base model without any customization. The service returns the following warning message: `Using non-customized default base model, because your custom {type} model has been built with a version of the base model that is no longer supported.` To resume using a custom model that is based on the unavailable model, you must first upgrade the model by using the appropriate `upgrade_model` method described previously.

12 June 2018

New features for applications hosted in Washington, DC, location

The following features are enabled for applications that are hosted in Washington, DC (**us-east**):

- The service now supports a new API authentication process. For more information, see the 30 October 2018 service update .
- The service now supports the `X-Watson-Metadata` header and the `DELETE /v1/user_data` method. For more information, see Information security.

15 May 2018

New features for applications hosted in Sydney location

The following features are enabled for applications that are hosted in Sydney (**au-syd**):

- The service now supports a new API authentication process. For more information, see the 30 October 2018 service update .
- The service now supports the `X-Watson-Metadata` header and the `DELETE /v1/user_data` method. For more information, see Information security.

26 March 2018

Language model customization now available for French broadband model

The service now supports language model customization for the French broadband language model, `fr-FR_BroadbandModel`. The French model is generally available (GA) for production use with language model customization.

- For more information about how the service parses corpora for French, see Parsing of Dutch, English, French, German, Italian, Portuguese, and Spanish.
- For more information about creating sounds-like pronunciations for custom words in French, see Guidelines for Dutch, French, German, Italian, Portuguese, and Spanish.

Updates to French, Korean, and Spanish models for improved speech recognition

The following models were updated for improved speech recognition:

- Korean narrowband model (`ko-KR_NarrowbandModel`)
- Spanish narrowband model (`es-ES_NarrowbandModel`)
- French broadband model (`fr-FR_BroadbandModel`)

By default, the service automatically uses the updated models for all recognition requests. If you have custom language or custom acoustic models that are based on either of these models, you must upgrade your custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

The `version` parameter renamed to `base_model_version`

The `version` parameter of the following methods is now named `base_model_version`:

- `/v1/recognize` for WebSocket requests
- `POST /v1/recognize` for sessionless HTTP requests
- `POST /v1/sessions` for session-based HTTP requests
- `POST /v1/recognitions` for asynchronous HTTP requests

The `base_model_version` parameter specifies the version of a base model that is to be used for speech recognition. For more information, see Using upgraded custom models for speech recognition and Making speech recognition requests with upgraded custom models.

New support for smart formatting for Spanish speech recognition

Smart formatting is now supported for Spanish as well as US English. For US English, the feature also now converts keyword strings into punctuation symbols for periods, commas, question marks, and exclamation points. For more information, see Smart formatting.

1 March 2018

Updates to French and Spanish broadband models for improved speech recognition

The French and Spanish broadband models, `fr-FR_BroadbandModel` and `es-ES_BroadbandModel`, have been updated for improved speech recognition. By default, the service automatically uses the updated models for all recognition requests. If you have custom language or custom acoustic models that are based on either of these models, you must upgrade your custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models. The section presents rules for upgrading custom models, the effects of upgrading, and approaches for using upgraded models.

1 February 2018

New Korean models

The service now offers language models for Korean: `ko-KR_BroadbandModel` for audio that is sampled at a minimum of 16 kHz, and `ko-KR_NarrowbandModel` for audio that is sampled at a minimum of 8 kHz. For more information, see Previous-generation languages and models.

For language model customization, the Korean models are generally available (GA) for production use; for acoustic model customization, they are beta functionality. For more information, see Language support for customization.

- For more information about how the service parses corpora for Korean, see Parsing of Korean.
- For more information about creating sounds-like pronunciations for custom words in Korean, see Guidelines for Korean.

14 December 2017

Language model customization now generally available

Language model customization and all associated parameters are now generally available (GA) for all supported languages: Japanese, Spanish, UK English, and US English.

Beta acoustic model customization now available for all languages

The service now supports acoustic model customization as beta functionality for all available languages. You can create custom acoustic models for broadband or narrowband models for all languages. For an introduction to customization, including acoustic model customization, see Understanding customization.

New `version` parameter for speech recognition

The various methods for making recognition requests now include a new `version` parameter that you can use to initiate requests that use either the older or upgraded versions of base and custom models. Although it is intended primarily for use with custom models that have been upgraded, the `version` parameter can also be used without custom models. For more information, see Making speech recognition requests with upgraded custom models.

Updates to US English models for improved speech recognition

The US English models, `en-US_BroadbandModel` and `en-US_NarrowbandModel`, have been updated for improved speech recognition. By default, the service automatically uses the updated models for all recognition requests. If you have custom language or custom acoustic models that are based on the US English models, you must upgrade your custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information about the procedure, see Upgrading custom models. The section presents rules for upgrading custom models, the effects of upgrading, and approaches for using upgraded models. Currently, the methods apply only to the new US English base models. But the same information will apply to upgrades of other base models as they become available.

Language model customization now available for UK English

The service now supports language model customization for the UK English models, `en-GB_BroadbandModel` and `en-GB_NarrowbandModel`. Although the service handles UK and US English corpora and custom words in a generally similar fashion, some important differences exist:

- For more information about how the service parses corpora for UK English, see Parsing of Dutch, English, French, German, Italian, Portuguese, and Spanish.
- For more information about creating sounds-like pronunciations for custom words in UK English, see Guidelines for English. Specifically, for UK English, you cannot use periods or dashes in sounds-like pronunciations.

2 October 2017

New beta acoustic model customization interface for US English, Japanese, and Spanish

The customization interface now offers acoustic model customization. You can create custom acoustic models that adapt the service's base models to your environment and speakers. You populate and train a custom acoustic model on audio that more closely matches the acoustic signature of the audio that you want to transcribe. You then use the custom acoustic model with recognition requests to increase the accuracy of speech recognition.

Custom acoustic models complement custom language models. You can train a custom acoustic model with a custom language model, and you can use both types of model during speech recognition. Acoustic model customization is a beta interface that is available only for US English, Japanese, and Spanish.

- For more information about the languages that are supported by the customization interface and the level of support that is available for each language, see Language support for customization.
- For more information about the service's customization interface, see Understanding customization.
- For more information about creating a custom acoustic model, see Creating a custom acoustic model.
- For more information about using a custom acoustic model, see Using a custom acoustic model for speech recognition.
- For more information about all methods of the customization interface, see the API & SDK reference.

New beta `customization_weight` parameter for custom language models

For language model customization, the service now includes a beta feature that sets an optional customization weight for a custom language model.

A customization weight specifies the relative weight to be given to words from a custom language model versus words from the service's base vocabulary. You can set a customization weight during both training and speech recognition. For more information, see Using customization weight.

Updates to Japanese broadband model for improved speech recognition

The `ja-JP_BroadbandModel` language model was upgraded to capture improvements in the base model. The upgrade does not affect existing custom models that are based on the model.

New `endianness` parameter for `audio/l16` audio format

The service now includes a parameter to specify the endianness of audio that is submitted in `audio/l16` (Linear 16-bit Pulse-Code Modulation (PCM)) format. In addition to specifying `rate` and `channels` parameters with the format, you can now also specify `big-endian` or `little-endian` with the `endianness` parameter. For more information, see audio/l16 format.

14 July 2017

New support for MP3 (MPEG) audio format

The service now supports the transcription of audio in the MP3 or Motion Picture Experts Group (MPEG) format. For more information, see audio/mp3 and audio/mpeg formats.

Beta language model customization now available for Spanish

The language model customization interface now supports Spanish as beta functionality. You can create a custom model based on either of the base Spanish language models: `es-ES_BroadbandModel` or `es-ES_NarrowbandModel`; for more information, see Creating a custom language model. Pricing for recognition requests that use Spanish custom language models is the same as for requests that use US English and Japanese models.

New `dialect` field for method that creates a custom language model

The JSON `CreateLanguageModel` object that you pass to the `POST /v1/customizations` method to create a new custom language model now includes a `dialect` field. The field specifies the dialect of the language that is to be used with the custom model. By default, the dialect matches the language of the base model. The parameter is meaningful only for Spanish models, for which the service can create a custom model that is suited for speech in one of the following dialects:

- `es-ES` for Castilian Spanish (the default)
- `es-LA` for Latin-American Spanish
- `es-US` for North-American (Mexican) Spanish

The `GET /v1/customizations` and `GET /v1/customizations/{customization_id}` methods of the customization interface include the dialect of a custom model in their output. For more information, see Creating a custom language model and Listing custom language models.

New names for UK English models

The names of the language models `en-UK_BroadbandModel` and `en-UK_NarrowbandModel` have been deprecated. The models are now available with the names `en-GB_BroadbandModel` and `en-GB_NarrowbandModel`.

The deprecated `en-UK_{model}` names continue to function, but the `GET /v1/models` method no longer returns the names in the list of available models. You can still query the names directly with the `GET /v1/models/{model_id}` method.

1 July 2017

Language model custom now generally available for US English and Japanese

The language model customization interface of the service is now generally available (GA) for both of its supported languages, US English and Japanese. IBM does not charge for creating, hosting, or managing custom language models. As described in the next bullet, IBM now charges an extra $0.03 (USD) per minute of audio for recognition requests that use custom models.

Updates to pricing plans for the service

IBM updated the pricing for the service by

- Eliminating the add-on price for using narrowband models

- Providing Graduated Tiered Pricing for high-volume customers
- Charging an additional $0.03 (USD) per minute of audio for recognition requests that use US English or Japanese custom language models

For more information about the pricing updates, see

- The Speech to Text service in the [IBM Cloud Catalog]
- The [Pricing FAQs]

Empty body no longer required for HTTP POST requests

You no longer need to pass an empty data object as the body for the following `POST` requests:

- `POST /v1/sessions`
- `POST /v1/register_callback`
- `POST /v1/customizations/{customization_id}/train`
- `POST /v1/customizations/{customization_id}/reset`
- `POST /v1/customizations/{customization_id}/upgrade_model`

For example, you now call the `POST /v1/sessions` method with `curl` as follows:

```
$ curl -X POST -u "{username}:{password}" \
--cookie-jar cookies.txt \
"{url}/v1/sessions"
```

You no longer need to pass the following `curl` option with the request: `--data "{}"`. If you experience any problems with one of these `POST` requests, try passing an empty data object with the body of the request. Passing an empty object does not change the nature or meaning of the request in any way.

22 May 2017

The `continuous` parameter removed from all methods

The `continuous` parameter is removed from all methods that initiate recognition requests. The service now transcribes an entire audio stream until it ends or times out, whichever occurs first. This behavior is equivalent to setting the former `continuous` parameter to `true`. By default, the service previously stopped transcription at the first half-second of non-speech (typically silence) if the parameter was omitted or set to `false`.

Existing applications that set the parameter to `true` will see no change in behavior. Applications that set the parameter to `false` or that relied on the default behavior will likely see a change. If a request specifies the parameter, the service now responds by returning a warning message for the unknown parameter:

```
"warnings": [
  "Unknown arguments: continuous."
]
```

The request succeeds despite the warning, and an existing session or WebSocket connection is not affected.

IBM removed the parameter to respond to overwhelming feedback from the developer community that specifying `continuous=false` added little value and could reduce overall transcription accuracy.

Sending audio required to avoid session timeout

It is no longer possible to avoid a session timeout without sending audio:

- When you use the WebSocket interface, the client can no longer keep a connection alive by sending a JSON text message with the `action` parameter set to `no-op`. Sending a `no-op` message does not generate an error, but it has no effect.
- When you use sessions with the HTTP interface, the client can no longer extend the session by sending a `GET /v1/sessions/{session_id}/recognize` request. The method still returns the status of an active session, but it does not keep the session active.

You can now do the following to keep a session alive:

- Set the `inactivity_timeout` parameter to `-1` to avoid the 30-second inactivity timeout.
- Send any audio data, including just silence, to the service to avoid the 30-second session timeout. You are charged for the duration of any data

that you send to the service, including the silence that you send to extend a session.

For more information, see [Timeouts](). Ideally, you would establish a session just before you obtain audio for transcription and maintain the session by sending audio at a rate that is close to real time. Also, make sure your application recovers gracefully from closed sessions or connections.

IBM removed this functionality to ensure that it continues to offer all users a best-in-class, low-latency speech recognition service.

10 April 2017

Speaker labels now supported for US English, Spanish, and Japanese

The service now supports the speaker labels feature for the following broadband models:

- US English broadband model (`en-US-BroadbandModel`)
- Spanish broadband model (`es-ES-BroadbandModel`)
- Japanese broadband model (`ja-JP_BroadbandModel`)

For more information, see Speaker labels.

New support for Web Media (WebM) audio format

The service now supports the Web Media (WebM) audio format with the Opus or Vorbis codec. The service now also supports the Ogg audio format with the Vorbis codec in addition to the Opus codec. For more information about supported audio formats, see audio/webm format.

New support for Cross-Origin Resource Sharing

The service now supports Cross-Origin Resource Sharing (CORS) to allow browser-based clients to call the service directly. For more information, see CORS support.

New method to unregister a callback URL with asynchronous HTTP interface

The asynchronous HTTP interface now offers a `POST /v1/unregister_callback` method that removes the registration for an allowlisted callback URL. For more information, see Unregistering a callback URL.

Defect fix: Eliminate timeouts for long audio with WebSocket interface

Defect fix: The WebSocket interface no longer times out for recognition requests for especially long audio files. You no longer need to request interim results with the JSON `start` message to avoid the timeout. (This issue was described in the update for 10 March 2016.)

New HTTP error codes

The following language model customization methods can now return the following HTTP error codes:

- The `DELETE /v1/customizations/{customization_id}` method now returns HTTP response code 401 if you attempt to delete a nonexistent custom model.
- The `DELETE /v1/customizations/{customization_id}/corpora/{corpus_name}` method now returns HTTP response code 400 if you attempt to delete a nonexistent corpus.

8 March 2017

Asynchronous HTTP interface is now generally available

The asynchronous HTTP interface is now generally available (GA). Prior to this date, it was beta functionality.

1 December 2016

New beta speaker labels feature

The service now offers a beta speaker labels feature for narrowband audio in US English, Spanish, or Japanese. The feature identifies which words were spoken by which speakers in a multi-person exchange. The sessionless, session-based, asynchronous, and WebSocket recognition methods each include a `speaker_labels` parameter that accepts a boolean value to indicate whether speaker labels are to be included in the response. For more information about the feature, see Speaker labels.

Beta language model customization now available for Japanese

The beta language model customization interface is now supported for Japanese in addition to US English. All methods of the interface support Japanese. For more information, see the following sections:

- For more information, see Creating a custom language model and Using a custom language model for speech recognition .
- For general and Japanese-specific considerations for adding a corpus text file, see Preparing a corpus text file and What happens when I add a corpus file?
- For Japanese-specific considerations when specifying the `sounds_like` field for a custom word, see Guidelines for Japanese.
- For more information about all methods of the customization interface, see the API & SDK reference.

New method for listing information about a corpus

The language model customization interface now includes a `GET /v1/customizations/{customization_id}/corpora/{corpus_name}` method that lists information about a specified corpus. The method is useful for monitoring the status of a request to add a corpus to a custom model. For more information, see Listing corpora for a custom language model.

New `count` field for methods that list words for custom language models

The JSON response that is returned by the `GET /v1/customizations/{customization_id}/words` and `GET /v1/customizations/{customization_id}/words/{word_name}` methods now includes a `count` field for each word. The field indicates the number of times the word is found across all corpora. If you add a custom word to a model before it is added by any corpora, the count begins at `1`. If the word is added from a corpus first and later modified, the count reflects only the number of times it is found in corpora. For more information, see Listing custom words from a custom language model.

For custom models that were created prior to the existence of the `count` field, the field always remains at `0`. To update the field for such models, add the model's corpora again and include the `allow_overwrite` parameter with the `POST /v1/customizations/{customization_id}/corpora/{corpus_name}` method.

New `sort` parameter for methods that list words for custom language models

The `GET /v1/customizations/{customization_id}/words` method now includes a `sort` query parameter that controls the order in which the words are to be listed. The parameter accepts two arguments, `alphabetical` or `count`, to indicate how the words are to be sorted. You can prepend an optional `+` or `-` to an argument to indicate whether the results are to be sorted in ascending or descending order. By default, the method displays the words in ascending alphabetical order. For more information, see Listing custom words from a custom language model.

For custom models created prior to the introduction of the `count` field, use of the `count` argument with the `sort` parameter is meaningless. Use the default `alphabetical` argument with such models.

New `error` field format for methods that list words for custom language models

The `error` field that can be returned as part of the JSON response from the `GET /v1/customizations/{customization_id}/words` and `GET /v1/customizations/{customization_id}/words/{word_name}` methods is now an array. If the service discovered one or more problems with a custom word's definition, the field lists each problem element from the definition and provides a message that describes the problem. For more information, see Listing custom words from a custom language model.

The `keywords_threshold` and `word_alternatives_threshold` parameters no longer accept a null value

The `keywords_threshold` and `word_alternatives_threshold` parameters of the recognition methods no longer accept a null value. To omit keywords and word alternatives from the response, omit the parameters. A specified value must be a float.

22 September 2016

New beta language model customization interface

The service now offers a new beta language model customization interface for US English. You can use the interface to tailor the service's base vocabulary and language models via the creation of custom language models that include domain-specific terminology. You can add custom words individually or have the service extract them from corpora. To use your custom models with the speech recognition methods that are offered by any of the service's interfaces, pass the `customization_id` query parameter. For more information, see

- Creating a custom language model
- Using a custom language model for speech recognition

- API & SDK reference

New support for `audio/mulaw` audio format

> The list of supported audio formats now includes `audio/mulaw`, which provides single-channel audio encoded using the u-law (or mu-law) data algorithm. When you use this format, you must also specify the sampling rate at which the audio is captured. For more information, see audio/mulaw format.

New `supported_features` identified when listing models

> The `GET /v1/models` and `GET /v1/models/{model_id}` methods now return a `supported_features` field as part of their output for each language model. The additional information describes whether the model supports customization. For more information, see the API & SDK reference.

30 June 2016

Beta asynchronous HTTP interface now supports all available languages

> The beta asynchronous HTTP interface now supports all languages that are supported by the service. The interface was previously available for US English only. For more information, see The asynchronous HTTP interface and the API & SDK reference.

23 June 2016

New beta asynchronous HTTP interface now available

> A beta asynchronous HTTP interface is now available. The interface provides full recognition capabilities for US English transcription via non-blocking HTTP calls. You can register callback URLs and provide user-specified secret strings to achieve authentication and data integrity with digital signatures. For more information, see The asynchronous HTTP interface and the API & SDK reference.

New beta `smart_formatting` parameter for speech recognition

> A beta smart formatting feature that converts dates, times, series of digits and numbers, phone numbers, currency values, and Internet addresses into more conventional representations in final transcripts. You enable the feature by setting the `smart_formatting` parameter to `true` on a recognition request. The feature is beta functionality that is available for US English only. For more information, see Smart formatting.

New French broadband model

> The list of supported models for speech recognition now includes `fr-FR_BroadbandModel` for audio in the French language that is sampled at a minimum of 16 kHz. For more information, see Previous-generation languages and models.

New support for `audio/basic` audio format

> The list of supported audio formats now includes `audio/basic`. The format provides single-channel audio that is encoded by using 8-bit u-law (or mu-law) data that is sampled at 8 kHz. For more information, see audio/basic format.

Speech recognition methods now return warnings for invalid parameters

> The various recognition methods can return a `warnings` response that includes messages about invalid query parameters or JSON fields that are included with a request. The format of the warnings changed. For example, `"warnings": "Unknown arguments: [u'{invalid_arg_1}', u'{invalid_arg_2}']."` is now `"warnings": "Unknown arguments: {invalid_arg_1}, {invalid_arg_2}."`

Empty body required for HTTP `POST` methods that pass no data

> For HTTP `POST` requests that do not otherwise pass data to the service, you must include an empty request body of the form `{}`. With the `curl` command, you use the `--data` option to pass the empty data.

10 March 2016

New maximum limits on audio transmitted for speech recognition

> Both forms of data transmission (one-shot delivery and streaming) now impose a size limit of 100 MB on the audio data, as does the WebSocket interface. Formerly, the one-shot approach had a maximum limit of 4 MB of data. For more information, see Audio transmission (for all interfaces) and Send audio and receive recognition results (for the WebSocket interface). The WebSocket section also discusses the maximum frame or

message size of 4 MB enforced by the WebSocket interface.

HTTP and WebSocket interfaces can now return warnings

The JSON response for a recognition request can now include an array of warning messages for invalid query parameters or JSON fields that are included with a request. Each element of the array is a string that describes the nature of the warning followed by an array of invalid argument strings. For example, `"warnings": [ "Unknown arguments: [u'{invalid_arg_1}', u'{invalid_arg_2}']." ]`. For more information, see the [API & SDK reference](#).

Beta Apple iOS SDK is deprecated

The beta *Watson Speech Software Development Kit (SDK) for the Apple® iOS operating system* is deprecated. Use the *Watson SDK for the Apple® iOS operating system* instead. The new SDK is available from the [ios-sdk repository](#) in the `watson-developer-cloud` namespace on GitHub.

WebSocket interface can produce delayed results

The WebSocket interface can take minutes to produce final results for a recognition request for an especially long audio file. For the WebSocket interface, the underlying TCP connection remains idle while the service prepares the response. Therefore, the connection can close due to a timeout. To avoid the timeout with the WebSocket interface, request interim results (`\"interim_results\": \"true\"`) in the JSON for the `start` message to initiate the request. You can discard the interim results if you do not need them. This issue will be resolved in a future update.

19 January 2016

New profanity filtering feature

The service was updated to include a new profanity filtering feature on January 19, 2016. By default, the service censors profanity from its transcription results for US English audio. For more information, see [Profanity filtering](#).

17 December 2015

New keyword spotting feature

The service now offers a keyword spotting feature. You can specify an array of keyword strings that are to be matched in the input audio. You must also specify a user-defined confidence level that a word must meet to be considered a match for a keyword. For more information, see [Keyword spotting](#). The keyword spotting feature is beta functionality.

New word alternatives feature

The service now offers a word alternatives feature. The feature returns alternative hypotheses for words in the input audio that meet a user-defined confidence level. For more information, see [Word alternatives](#). The word alternatives feature is beta functionality.

New UK English and Arabic models

The service supports more languages with its transcription models: `en-UK_BroadbandModel` and `en-UK_NarrowbandModel` for UK English, and `ar-AR_BroadbandModel` for Modern Standard Arabic. For more information, see [Previous-generation languages and models](#).

New `session_closed` field for session-based methods

In the JSON responses that it returns for errors with session-based methods, the service now also includes a new `session_closed` field. The field is set to `true` if the session is closed as a result of the error. For more information about possible return codes for any method, see the [API & SDK reference](#).

HTTP platform timeout no longer applies

HTTP recognition requests are no longer subject to a 10-minute platform timeout. The service now keeps the connection alive by sending a space character in the response JSON object every 20 seconds while recognition is ongoing. For more information, see [Timeouts](#).

Rate limiting with curl command is no longer needed

When you use the `curl` command to transcribe audio with the service, you no longer need to use the `--limit-rate` option to transfer data at a rate no faster than 40,000 bytes per second.

Changes to HTTP error codes

The service no longer returns HTTP status code 490 for the session-based HTTP methods `GET /v1/sessions/{session_id}/observe_result` and `POST /v1/sessions/{session_id}/recognize`. The service now responds with HTTP status code 400 instead.

21 September 2015

New mobile SDKs available

Two new beta mobile SDKs are available for the speech services. The SDKs enable mobile applications to interact with both the Speech to Text and Text to Speech services.

- The *Watson Speech SDK for the Google Android™ platform* supports streaming audio to the Speech to Text service in real time and receiving a transcript of the audio as you speak. The project includes an example application that showcases interaction with both of the speech services. The SDK is available from the speech-android-sdk repository in the `watson-developer-cloud` namespace on GitHub.
- The *Watson Speech SDK for the Apple® iOS operating system* supports streaming audio to the Speech to Text service and receiving a transcript of the audio in response. The SDK is available from the speech-ios-sdk repository in the `watson-developer-cloud` namespace on GitHub.

Both SDKs support authenticating with the speech services by using either your IBM Cloud service credentials or an authentication token. Because the SDKs are beta, they are subject to change in the future.

New Brazilian Portuguese and Mandarin Chinese models

The service supports two new languages, Brazilian Portuguese and Mandarin Chinese, with the following models:

- Brazilian Portuguese broadband model (`pt-BR_BroadbandModel`)
- Brazilian Portuguese narrowband model (`pt-BR_NarrowbandModel`)
- Mandarin Chinese broadband model (`zh-CN_BroadbandModel`)
- Mandarin Chinese narrowband model (`zh-CN_NarrowbandModel`)

For more information, see Previous-generation languages and models.

New support for `audio/ogg;codecs=opus` audio format

The HTTP `POST` requests `/v1/sessions/{session_id}/recognize` and `/v1/recognize` , as well as the WebSocket `/v1/recognize` request, support transcription of a new media type: `audio/ogg;codecs=opus` for Ogg format files that use the Opus codec. In addition, the `audio/wav` format for the methods now supports any encoding. The restriction about the use of linear PCM encoding is removed. For more information, see audio/ogg format.

New `sequence_id` parameter for long polling of sessions

The service now supports overcoming timeouts when you transcribe long audio files with the HTTP interface. When you use sessions, you can employ a long polling pattern by specifying sequence IDs with the `GET /v1/sessions/{session_id}/observe_result` and `POST /v1/sessions/{session_id}/recognize` methods for long-running recognition tasks. By using the new `sequence_id` parameter of these methods, you can request results before, during, or after you submit a recognition request.

New capitalization feature for US English transcription

For the US English language models, `en_US_BroadbandModel` and `en_US_NarrowbandModel` , the service now correctly capitalizes many proper nouns. For example, the service would return new text that reads "Barack Obama graduated from Columbia University" instead of "barack obama graduated from columbia university". This change might be of interest to you if your application is sensitive in any way to the case of proper nouns.

New HTTP error code

The HTTP `DELETE /v1/sessions/{session_id}` request does not return status code 415 "Unsupported Media Type". This return code is removed from the documentation for the method.

1 July 2015

The Speech to Text service is now generally available

The service moved from beta to general availability (GA) on July 1, 2015. The following differences exist between the beta and GA versions of the Speech to Text APIs. The GA release requires that users upgrade to the new version of the service.

The GA version of the HTTP API is compatible with the beta version. You need to change your existing application code only if you explicitly specified a model name. For example, the sample code available for the service from GitHub included the following line of code in the file `demo.js` :

```
model: 'WatsonModel'
```

This line specified the default model `WatsonModel`, for the beta version of the service. If your application also specified this model, you need to change it to use one of the new models that are supported by the GA version. For more information, see the next bullet.

New token-based programming model

The service now supports a new programming model for direct interaction between a client and the service over a WebSocket connection. By using this model, a client can obtain an authentication token for communicating directly with the service. The token bypasses the need for a server-side proxy application in IBM Cloud to call the service on the client's behalf. Tokens are the preferred means for clients to interact with the service.

The service continues to support the old programming model that relied on a server-side proxy to relay audio and messages between the client and the service. But the new model is more efficient and provides higher throughput.

New `model` parameter for speech recognition

The `POST /v1/sessions` and `POST /v1/recognize` methods, along with the WebSocket `/v1/recognize` method, now support a `model` query parameter. You use the parameter to specify information about the audio:

- The language: *English*, *Japanese*, or *Spanish*
- The minimum sampling rate: *broadband* (16 kHz) or *narrowband* (8 kHz)

For more information, see Previous-generation languages and models.

New `inactivity_timeout` parameter for speech recognition

The `inactivity_timeout` parameter sets the timeout value in seconds after which the service closes the connection if it detects silence (no speech) in streaming mode. By default, the service terminates the session after 30 seconds of silence. The `POST /v1/recognize` and WebSocket `/v1/recognize` methods support the parameter. For more information, see Timeouts.

New `max_alternatives` parameter for speech recognition

The `max_alternatives` parameter instructs the service to return the *n*-best alternative hypotheses for the audio transcription. The `POST /v1/recognize` and WebSocket `/v1/recognize` methods support the parameter. For more information, see Maximum alternatives.

New `word_confidence` parameter for speech recognition

The `word_confidence` parameter instructs the service to return a confidence score for each word of the transcription. The `POST /v1/recognize` and WebSocket `/v1/recognize` methods support the parameter. For more information, see Word confidence.

New `timestamps` parameter for speech recognition

The `timestamps` parameter instructs the service to return the beginning and ending time relative to the start of the audio for each word of the transcription. The `POST /v1/recognize` and WebSocket `/v1/recognize` methods support the parameter. For more information, see Word timestamps.

Renamed sessions method for observing results

The `GET /v1/sessions/{session_id}/observeResult` method is now named `GET /v1/sessions/{session_id}/observe_result`. The name `observeResult` is still supported for backward compatibility.

New support for Waveform Audio File (WAV) audio format

The `Content-Type` header of the `recognize` methods now supports `audio/wav` for Waveform Audio File (WAV) files, in addition to `audio/flac` and `audio/l16`. For more information, see audio/wav format.

Limits on maximum amount of audio for speech recognition

The service now has a limit of 100 MB of data per session in streaming mode. You can specify streaming mode by specifying the value `chunked` with the header `Transfer-Encoding`. One-shot delivery of an audio file still imposes a size limit of 4 MB on the data that is sent. For more information, see Audio transmission.

New header to opt out of contributing to service improvements

The `GET /v1/sessions/{session_id}/observe_result`, `POST /v1/sessions/{session_id}/recognize`, and `POST /v1/recognize` methods now include the header parameter `X-WDC-PL-OPT-OUT` to control whether the service uses the audio and transcription data from a request to improve future results. The WebSocket interface includes an equivalent query parameter. Specify a value of `1` to prevent the service from using the audio and transcription results. The parameter applies only to the current request. The new header replaces the `X-logging` header from the beta API. See Controlling request logging for Watson services.

Changes to HTTP error codes

The service can now respond with the following HTTP error codes:

- For the `/v1/models`, `/v1/models/{model_id}`, `/v1/sessions`, `/v1/sessions/{session_id}`, `/v1/sessions/{session_id}/observe_result`, `/v1/sessions/{session_id}/recognize`, and `/v1/recognize` methods, error code 415 ("Unsupported Media Type") is added.
- For `POST` and `GET` requests to the `/v1/sessions/{session_id}/recognize` method, the following error codes are modified:
  - Error code 404 ("Session_id not found") has a more descriptive message (`POST` and `GET`).
  - Error code 503 ("Session is already processing a request. Concurrent requests are not allowed on the same session. Session remains alive after this error.") has a more descriptive message (`POST` only).
  - For HTTP `POST` requests to the `/v1/sessions` and `/v1/recognize` methods, error code 503 ("Service Unavailable") can be returned. The error code can also be returned when you create a WebSocket connection with the `/v1/recognize` method.

Release notes for Speech to Text for IBM Cloud Pak for Data

IBM Cloud Pak for Data

The following features and changes were included for each release and update of installed or on-premises instances of IBM Watson® Speech to Text for IBM Cloud Pak for Data. Unless otherwise noted, all changes are compatible with earlier releases and are automatically and transparently available to all new and existing applications.

For information about known limitations of the service, see Known limitations.

Note: For information about releases and updates of the service for IBM Cloud, see Release notes for Speech to Text for IBM Cloud.

2 May 2023 (Version 4.6.5)

Version 4.6.5 is now available

Speech to Text for IBM Cloud Pak for Data version 4.6.5 is now available. This version supports IBM Cloud Pak for Data version 4.6.x and Red Hat OpenShift versions 4.10 and 4.12. For more information, see Watson Speech services on IBM Cloud Pak for Data.

New Japanese next-generation telephony model

The service now offers a next-generation telephony model for Japanese: `ja-JP_Telephony`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Improved language model customization for next-generation English and Japanese models

The service now provides improved language model customization for next-generation English and Japanese models:

- `en-AU_Multimedia`
- `en-AU_Telephony`
- `en-IN_Telephony`
- `en-GB_Multimedia`
- `en-GB_Telephony`
- `en-US_Multimedia`
- `en-US_Telephony`
- `ja-JP_Multimedia`
- `ja-JP_Telephony`

Visible improvements to the models: The new technology improves the default behavior of the new English and Japanese models. Among other changes, the new technology optimizes the default behavior for the following parameters:

- The default `customization_weight` for custom models that are based on the new versions of these models changes from `0.2` to `0.1`.
- The default `character_insertion_bias` for custom models that are based on the new versions of these models remains `0.0`, but the models have changed in a manner that makes use of the parameter for speech recognition less necessary.

Upgrading to the new models: To take advantage of the improved technology, you must upgrade any custom language models that are based on the new models. To upgrade to the new version of one of these base models, do the following:

1. Change your custom model by adding or modifying a custom word, corpus, or grammar that the model contains. Any change that you make moves the model to the `ready` state.
2. Use the `POST /v1/customizations/{customization_id}/train` method to retrain the model. Retraining upgrades the custom model to the new technology and moves the model to the `available` state.

   Known issue: At this time, you cannot use the `POST /v1/customizations/{customization_id}/upgrade_model` method to upgrade a custom model to one of the new base models. This issue will be addressed in a future release.

Using the new models: Following the upgrade to the new base model, you are advised to evaluate the performance of the upgraded custom model by paying special attention to the `customization_weight` and `character_insertion_bias` parameters for speech recognition. When you retrain your custom model:

- The custom model uses the new default `customization_weight` of `0.1` for your custom model. A non-default `customization_weight` that you had associated with your custom model is removed.
- The custom model might no longer require use of the `character_insertion_bias` parameter for optimal speech recognition.

Improvements to language model customization render these parameters less important for high-quality speech recognition:

- If you use the default values for these parameters, continue to do so after the upgrade. The default values will likely continue to offer the best results for speech recognition.
- If you specify non-default values for these parameters, experiment with the default values following upgrade. Your custom model might work well for speech recognition with the default values.

If you feel that using different values for these parameters might improve speech recognition with your custom model, experiment with incremental changes to determine whether the parameters are needed to improve speech recognition.

Note: At this time, the improvements to language model customization apply only to custom models that are based on the next-generation English or Japanese base language models listed earlier. Over time, the improvements will be made available for other next-generation language models.

More information: For more information about upgrading and about speech recognition with these parameters, see

- Improved language model customization for next-generation models
- Upgrading custom models
- Using customization weight
- Character insertion bias

New environment variable for Speech services custom resource

The documentation now includes instructions to create an environment variable named `${CUSTOM_RESOURCE_SPEECH}`. You append the new variable to the `cpd_vars.sh` script, and source the script to use the variable in your environment. For more information, see *Information you need to complete this task* in Installing Watson Speech services, or refer to any of the upgrade topics for the Speech services.

Defect fix: The Swedish telephony and Italian multimedia models are now available

Defect fix: The Swedish telephony (`sv-SE_Telephony`) and Italian multimedia (`it-IT_Multimedia`) models are now available for installation. Previously, they were not available.

Defect fix: Improved training time for next-generation custom language models

Defect fix: Training time for next-generation custom language models is now significantly improved. Previously, training time took much longer than necessary, as reported for training of Japanese custom language models. The problem was corrected by an internal fix.

Defect fix: Grammar files now handle strings of digits correctly

Defect fix: When grammars are used, the service now handles longer strings of digits correctly. Previously, it was failing to complete recognition or returning incorrect results.

Defect fix: Dynamically generated grammar files now work properly

Defect fix: Dynamically generated grammar files now work properly. Previously, dynamic grammar files could cause internal failures, as reported for integration of Speech to Text with IBM® watsonx™ Assistant. The problem was corrected by an internal fix.

Defect fix: Smart formatting for US English dates is now correct

Defect fix: Smart formatting now correctly includes days of the week and dates when both are present in the spoken audio, for example, `Tuesday`

`February 28`. Previously, in some cases the day of the week was omitted and the date was presented incorrectly. Note that smart formatting is beta functionality.

Defect fix: Update documentation for speech hesitation words for next-generation models

Defect fix: Documentation for speech hesitation words for next-generation models has been updated. More details are provided about US English and Japanese hesitation words. Next-generation models include the actual hesitation words in transcription results, unlike previous-generation models, which include only hesitation markers. For more information, see Speech hesitations and hesitation markers.

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Python (CVE-2020-10735)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to phishing attacks in Python (CVE-2021-28861)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Pypa Setuptools (CVE-2022-40897)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a sensitive information exposure in systemd (CVE-2022-4415)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Python (CVE-2022-45061)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in Libksba (CVE-2022-47629)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in GNU Tar (CVE-2022-48303)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in FasterXML jackson-databind (CVE-2022-42003)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in Perl (CVE-2020-10878)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in Apache Tomcat (CVE-2022-45143)
- CVE-2020-10543: Publication of the security bulletin is pending.

29 March 2023 (Version 4.6.4)

Version 4.6.4 is now available

Speech to Text for IBM Cloud Pak for Data version 4.6.4 is now available. This version supports IBM Cloud Pak for Data version 4.6.x and Red Hat OpenShift versions 4.10 and 4.12. For more information, see Watson Speech services on IBM Cloud Pak for Data.

Important: Back up your data before upgrading to version 4.6.3 or 4.6.4

Important: Before upgrading to Watson Speech services version 4.6.3 or 4.6.4, you must make a backup of your data. Preserve the backup in a safe location. For more information about backing up your Watson Speech services data, see *Backing up and restoring Watson Speech services data* in Administering Watson Speech services. That topic also includes information about restoring your data if that becomes necessary.

Known issue: The Swedish telephony and Italian multimedia models are not yet available

Known issue: The Swedish telephony (`sv-SE_Telephony`) and Italian multimedia (`it-IT_Multimedia`) models are not yet available. They will be made available with version 4.6.5.

Defect fix: You can now change the installed models and voices with the advanced installation options

Defect fix: During installation, you can now specify different models or voices with the advanced installation options of the command-line interface. Previously, the service always installed the default models and voices. The limitation continues to apply for Watson Speech services versions 4.6.0, 4.6.2, and 4.6.3. For information about installing models and voices, see *Specifying additional installation options* in Installing Watson Speech services.

Setting load balancer timeouts

Watson Speech services require that you change the load balancer timeout settings for both the server and client to 300 seconds. These settings ensure that long-running speech recognition requests, those with long or difficult audio, have sufficient time to complete. For more information, see

Information you need to complete this task in Installing Watson Speech services.

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to cross-site scripting in GNOME libxml2 (CVE-2016-3709
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in SQlite (CVE-2020-35525)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in Amazon AWS S3 Crypto SDK for GoLang (CVE-2020-8912)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to elevated system privileges in the Red Hat Build of OpenJDK (CVE-2021-20264)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to an arbitrary code execution in e2fsprogs (CVE-2022-1304)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to errors in TrustCor (CVE-2022-23491)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in GnuTLS (CVE-2022-2509)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to an arbitrary code execution in systemd (CVE-2022-2526)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to sensitive information exposure in AWS SDK for Go (CVE-2022-2582)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to denial of service in cURL libcurl (CVE-2022-32206)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a man-in-the-middle attack in cURL libcurl (CVE-2022-32208)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to spoofing attacks in GnuPG (CVE-2022-34903)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in SQLite (CVE-2022-35737)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in zlib (CVE-2022-37434)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in systemd (CVE-2022-3821)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to an arbitrary code execution in Gnome libxml2 (CVE-2022-40303)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to an arbitrary code execution in Gnome libxml2 (CVE-2022-40304)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Python Charmers Future (CVE-2022-40899)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in Golang Go (CVE-2022-41716)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-41717)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Freedesktop D-Bus (CVE-2022-42010)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Freedesktop D-Bus (CVE-2022-42011)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Freedesktop D-Bus (CVE-2022-42012)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in MIT krb5 (CVE-2022-42898)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in libexpat (CVE-2022-43680)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to an arbitrary commands execution in Python (CVE-2015-20107)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in SQlite (CVE-

2020-35527)

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in GNU Libtasn1 (CVE-2021-46848)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in Git (CVE-2022-23521)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in GnuPG Libksba (CVE-2022-3515)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to an arbitrary code execution in libexpat (CVE-2022-40674)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in Git (CVE-2022-41903)

23 February 2023 (Version 4.6.3)

Version 4.6.3 is now available

Speech to Text for IBM Cloud Pak for Data version 4.6.3 is now available. This version supports IBM Cloud Pak for Data version 4.6.x and Red Hat OpenShift version 4.10. Red Hat OpenShift version 4.8 is no longer supported. For more information, see Watson Speech services on IBM Cloud Pak for Data.

Important: All previous-generation models are deprecated and will reach end of service on 31 July 2023

Important: All previous-generation models are deprecated and will reach end of service effective **31 July 2023**. On that date, all previous-generation models will be removed from the service and the documentation. The previous deprecation date was 3 March 2023. The new date allows users more time to migrate to the appropriate next-generation models. But users must migrate to the equivalent next-generation model by 31 July 2023.

Most previous-generation models were deprecated on 15 March 2022. Previously, the Arabic and Japanese models were not deprecated. Deprecation now applies to *all* previous-generation models.

- For more information about the next-generation models to which you can migrate from each of the deprecated models, see Previous-generation languages and models
- For more information about migrating from previous-generation to next-generation models, see Migrating to next-generation models.
- For more information about all next-generation models, see Next-generation languages and models

Note: When the previous-generation `en-US_BroadbandModel` is removed from service, the next-generation `en-US_Multimedia` model will become the default model for speech recognition requests.

Known issue: You cannot change the installed models and voices with the advanced installation options

Known issue: You currently cannot specify different models or voices with the advanced installation options. The service always installs the default models and voices. For information about changing the models after installation, see *Updating models and voices for your Watson Speech services* in the *Administration* topic of Watson Speech services on IBM Cloud Pak for Data.

Known issue: Upgrade to version 4.6.3 can fail to complete

Known issue: When upgrading to version 4.6.3, the MinIO backup job can fail to be deleted upon completion. If this happens, the solution is to delete the job, after which the upgrade proceeds normally. Perform the following steps to resolve the problem.

1. To determine whether the MinIO backup job remains undeleted, issue the following command:

   ```
   $ oc get job --namespace {${PROJECT_CPD_INSTANCE} | grep speech-cr-ibm-minio-backup
   ```

   The MinIO job that is not deleted is identified by an entry of the following form:

   ```
   speech-cr-ibm-minio-backup   1/1   3m25s   1d
   ```

2. To delete the MinIO backup job, issue the following command:

   ```
   $ oc delete job speech-cr-ibm-minio-backup --namespace ${PROJECT_CPD_INSTANCE}
   ```

Once the backup job is deleted, upgrade continues and completes.

Defect fix: Update French Canadian next-generation telephony model (upgrade required)

> **Defect fix:** The French Canadian next-generation telephony model, `fr-CA_Telephony`, was updated to address an internal inconsistency that could cause an error during speech recognition. *You need to upgrade any custom models that are based on the `fr-CA_Telephony` model.* For more information about upgrading custom models, see
>
> - Upgrading custom models
> - Using upgraded custom models for speech recognition

Defect fix: The next-generation Brazilian Portuguese multimedia model is now available

> **Defect fix:** The next-generation Brazilian Portuguese multimedia model is now available for Speech to Text for IBM Cloud Pak for Data. Previously, the model was unavailable.

Adding words directly to custom models that are based on next-generation models increases the training time

> Adding custom words directly to a custom model that is based on a next-generation model causes training of a model to take a few minutes longer than it otherwise would. If you are training a model with custom words that you added by using the `POST /v1/customizations/{customization_id}/words` or `PUT /v1/customizations/{customization_id}/words/{word_name}` method, allow for some minutes of extra training time for the model. For more information, see
>
> - Add words to the custom language model
> - Monitoring the train model request

Additional information about working with service instances

> The documentation now includes information about creating a service instance with the command-line interface (`cpl-cli`) and about managing service instances. For more information, see the following topics of Watson Speech services on IBM Cloud Pak for Data :
>
> - *Creating a Watson Speech services instance* under *Post-installation setup*
> - *Managing your Watson Speech services instances* under *Administering*

Security vulnerability addressed

> The following security vulnerability has been fixed:
>
> - Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to denial of service in Pypa Setuptools (CVE-2022-40897)

30 January 2023 (Version 4.6.2)

Version 4.6.2 is now available

> Speech to Text for IBM Cloud Pak for Data version 4.6.2 is now available. This version supports IBM Cloud Pak for Data version 4.6.x and Red Hat OpenShift versions 4.8 and 4.10. For more information, see Watson Speech services on IBM Cloud Pak for Data .

The custom resource now includes a new `fileStorageClass` property

> The custom resource for the Watson Speech services now includes a `fileStorageClass` property in addition to the existing `blockStorageClass` property. You specify both block and file storage classes when you install or upgrade a service. During upgrade from a previous version, the new property is added automatically to the custom resource by the `--file_storage_class` option on `cli manage apply-cr` command.
>
> For more information about the available block and file storage classes you use with each of the supported storage solutions, see the table of *Storage requirements* under *Information you need to complete this task* on the page "Installing Watson Speech services" in Watson Speech services on IBM Cloud Pak for Data.

Additional information about provisioning a service instance

> The documentation now includes information about creating a service instance programmatically. It also includes examples of listing service instances and deleting a service instance. For more information, see *Creating a Watson Speech services instance* in the *Post-installation setup* documentation in Watson Speech services on IBM Cloud Pak for Data .

Server-side encryption is enabled for the MinIO datastore

> The Speech services have now enabled server-side encryption for object storage in the MinIO datastore. No action is required on your part.

Change to audit webhooks

The Speech services have now removed the audit webhook dependency. The services now write audit events directly to the server. After upgrading to version 4.6.2, some webhook resources might remain until all services can remove the dependency. The remaining resources will be removed in a future release. No action is required on your part.

New Netherlands Dutch next-generation multimedia model

The service now offers a next-generation multimedia model for Netherlands Dutch: `nl-NL_Multimedia`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

New Swedish next-generation telephony model

The service now offers a next-generation telephony model for Swedish: `sv-SE_Telephony`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Updates to English next-generation telephony models

The English next-generation telephony models have been updated for improved speech recognition:

- `en-AU_Telephony`
- `en-GB_Telephony`
- `en-IN_Telephony`
- `en-US_Telephony`

All of these models continue to support low latency. You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

The `max_alternatives` parameter is now available for use with next-generation models

The `max_alternatives` parameter is now available for use with all next-generation models. The parameter is generally available for all next-generation models. For more information, see Maximum alternatives.

Defect fix: Allow use of both `max_alternatives` and `end_of_phrase_silence_time` parameters with next-generation models

Defect fix: When you use both the `max_alternatives` and `end_of_phrase_silence_time` parameters in the same request with next-generation models, the service now returns multiple alternative transcripts while also respecting the indicated pause interval. Previously, use of the two parameters in a single request generated a failure. (Use of the `max_alternatives` parameter with next-generation models was previously available as an experimental feature to a limited number of customers.)

Defect fix: Update to Japanese next-generation multimedia model (upgrade required)

Defect fix: The Japanese next-generation multimedia model, `ja-JP_Multimedia`, was updated to address an internal inconsistency that could cause an error during speech recognition with low latency. *You need to upgrade any custom models that are based on the* `ja-JP_Multimedia` *model.* For more information about upgrading custom models, see

- Upgrading custom models
- Using upgraded custom models for speech recognition

Defect fix: Add documentation guidelines for creating Japanese sounds-likes based on next-generation models

Defect fix: In sounds-likes for Japanese custom language models that are based on next-generation models, the character-sequence `ウー` is ambiguous in some left contexts. Do not use characters (syllables) that end with the phoneme `/o/`, such as `ロ` and `ト`. In such cases, use `ウウ` or just `ウ` instead of `ウー`. For example, use `ロウウマン` or `ロウマン` instead of `ロウーマン`. For more information, see Guidelines for Japanese.

Defect fix: Correct use of `display_as` field in transcription results

Defect fix: For language model customization with next-generation models, the value of the `display_as` field for a custom word now appears in all transcripts. Previously, the value of the `word` field sometimes appeared in transcription results.

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to issues in OpenSSL (CVE-2022-1434, CVE-2022-1343, CVE-2022-1292, CVE-2022-1473)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary command execution in OpenSSL (CVE-2022-2068)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in protobuf (CVE-2022-1941)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a buffer overflow in GNU glibc (CVE-2021-3999)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security bypass in GNU gzip (CVE-2022-1271)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-27664)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-2879)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to query parameter smuggling in Golang Go (CVE-2022-2880)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-32189)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-41715)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to information exposure in OpenSSL (CVE-2022-2097)

30 November 2022 (Version 4.6.0)

Version 4.6.0 is now available

Speech to Text for IBM Cloud Pak for Data version 4.6.0 is now available. This version supports IBM Cloud Pak for Data version 4.6.x and Red Hat OpenShift versions 4.8 and 4.10. For more information, see Watson Speech services on IBM Cloud Pak for Data .

Amazon Web Services (AWS) is now supported

Watson Speech services for IBM Cloud Pak for Data are now supported on Amazon Web Services™ (AWS™). The services support Amazon Elastic Block Store, which you specify by setting the `blockStorageClass` property of the Speech services custom resource to `gp2-csi` or `gp3-csi`.

New storage classes are now supported

Watson Speech services for IBM Cloud Pak for Data now support two additional storage classes:

- IBM Cloud Block Storage (`ibmc-block-gold`)
- NetApp Trident (`ontap-nas`)

You specify the storage class with the `blockStorageClass` property of the Speech services custom resource. For more information about all supported storage classes, see the following topics in Watson Speech services on IBM Cloud Pak for Data :

- *Before you begin* in *Installing Watson Speech services*
- *Specifying a storage class* in *Using the Watson Speech services custom resource*

Known issue: Some Watson Speech services pods do not have annotations that are used for scheduling

Known issue: Some Watson Speech services pods are missing the `cloudpakInstanceId` annotation. If you use the IBM Cloud Pak for Data scheduling service, any Watson Speech services pods without the `cloudpakInstanceId` annotation are

- Scheduled by the default Kubernetes scheduler rather than the scheduling service
- Not included in the quota enforcement

Monitoring of the PostgreSQL datastore is now available

You can now enable monitoring of the PostgreSQL datastore to receive updates on its usage and status by the Watson Speech services. The events can be consumed by Prometheus monitoring software or whatever application you use for monitoring. By enabling monitoring for user-defined projects in addition to the default platform monitoring, you can monitor your own projects with the Red Hat® OpenShift® Container Platform monitoring stack. This capability includes an additional property, `spec.global.datastores.postgressql.enablePodMonitor`, in the Speech services custom resource.

For more information, see the topic *Monitoring the PostgreSQL datastore for Watson Speech services* in the *Administering* section of Watson Speech services on IBM Cloud Pak for Data.

Defect fix: PostgreSQL datastore is no longer installed if only runtime microservices are enabled

Defect fix: The PostgreSQL datastore is no longer installed if only the runtime microservices are enabled. The datastore is now installed only if at least one of the `sttAsync`, `sttCustomization`, or `ttsCustomization` microservices is installed. PostgreSQL is not uninstalled if at a later date these microservices are disabled.

Prior to version 4.6.0, PostgreSQL was always installed with the Speech services. If you are an existing customer who used only the runtime microservices of the Speech services prior to version 4.6.0, PostgreSQL remains installed but is not used. In this case, installation of PostgreSQL persists across upgrades.

The MinIO datastore is always installed because the runtime microservices depend on it. The RabbitMQ datastore is installed only if the `sttAsync` microservice is installed.

For more information, see *Datastore properties* in *Using the Watson Speech services custom resource* in Watson Speech services on IBM Cloud Pak for Data.

Defect fix: Creation of a Network Policy is no longer necessary for the PostgreSQL operator to monitor its operands

Defect fix: For version 4.6.0, it is not necessary to create a Network Policy to allow the PostgreSQL operator to monitor its operands, as described in the 10 November 2022 (Versions 4.0.x and 4.5.x) service update. As of version 4.6.0, the service handles this situation automatically.

Defect fix: Some next-generation models were updated to improve low-latency response time

Defect fix: The following next-generation models were updated to improve their response time when the `low_latency` parameter is used:

- `en-IN_Telephony`
- `hi-IN_Telephony`
- `it-IT_Multimedia`
- `nl-NL_Telephony`

Previously, these models did not return recognition results as quickly as expected when the `low_latency` parameter was used. You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

Defect fix: Improve custom model naming documentation

Defect fix: The documentation now provides detailed rules for naming custom language models and custom acoustic models. For more information, see

- Create a custom language model
- Create a custom acoustic model
- API & SDK reference

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a cross-configuration attack against OpenPGP (CVE-2021-40528)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in PCRE2 (CVE-2022-1586)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in Vim (CVE-2022-1621)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a buffer overflow in Vim (CVE-2022-1629)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in Vim (CVE-2022-1785, CVE-2022-1897, CVE-2022-1927)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in cURL libcurl (CVE-2022-22576)

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to credential exposure in cURL libcurl (CVE-2022-27774)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to data information exposure in cURL libcurl (CVE-2022-27776)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in cURL libcurl (CVE-2022-27782)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in GNOME libxml2 (CVE-2022-29824)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a SQL injection in PostgreSQL (CVE-2022-31197)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in libexpat (CVE-2022-25313)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in libexpat (CVE-2022-25314)

10 November 2022 (Versions 4.0.x and 4.5.x)

Known issue: Updated Network Policy needed for PostgreSQL operator

Known issue: For Speech services version 4.0.x (not including version 4.0.0) and 4.5.x, if the PostgreSQL operator and the Speech services are installed in different namespaces, the PostgreSQL operator is not able to monitor the PostgreSQL operands for the Speech services. The operator is prevented from monitoring the operands by the Network Policy that is in place for the Speech services.

This problem does not prevent the PostgreSQL cluster from functioning properly. The cluster remains active and fully functional. However, the operator is not able to update the operands when you upgrade to new versions of the Speech services.

The solution for the problem is to create an additional Network Policy for the PostgreSQL operator, as shown in the following steps. You can perform the steps regardless of whether the PostgreSQL operator is installed in the same namespace as the Speech services or in a different namespace.

1. Log in as an administrator of the Red Hat® OpenShift® project where the Speech services are installed.
2. Enter the following command to update the Network Policy for the Speech services:

```
$ cat << EOF | oc apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  labels:
    app.kubernetes.io/component: stt
    app.kubernetes.io/instance: {{ <custom-resource-name> }}
    app.kubernetes.io/name: speech-to-text
    release: {{ <custom-resource-name> }}
  name: <custom-resource-name>-postgres-network-policy
  namespace: {{ <cpd-instance-namespace> }}
spec:
  ingress:
  - from:
    - namespaceSelector: {}
      podSelector:
        matchLabels:
          app.kubernetes.io/name: cloud-native-postgresql
EOF
```

   where

   - `<custom-resource-name>` is the name of the Speech services custom resource. The recommended name for version 4.0.x is `speech-prod-cr`; the recommended name for version 4.5.x is `speech-cr`.
   - `<cpd-instance-name>` is the name of the project (namespace) in which the Speech services are installed. The documentation uses the environment variable `${PROJECT_CPD_INSTANCE}` to identity the namespace.

3. To verify that the updated Network Policy allows the operator to monitor the operands and that the PostgreSQL cluster is in a healthy state, enter the following command, where `<custom-resource-name>` and `<cpd-instance-name>` are the values you used in the previous step:

```
oc -get cluster {{ <custom-resource-name> }}-postgres -n {{ <cpd-instance-namespace> }}
```

If the PostgreSQL cluster is functioning properly, the command produces output similar to the following:

```
NAME                AGE   INSTANCES   READY   STATUS                     PRIMARY
speech-cr-postgres  14d   3           3       Cluster in healthy state   speech-cr-postgres-1
```

These steps do not cause operator to update the operands to the latest versions. However, the operands are upgraded as expected when you next upgrade the Speech services software.

13 October 2022 (Version 4.5.3)

Version 4.5.3 is now available

Speech to Text for IBM Cloud Pak for Data version 4.5.3 is now available. This version supports IBM Cloud Pak for Data version 4.5.x and Red Hat OpenShift versions 4.6, 4.8, and 4.10. For more information, see Watson Speech services on IBM Cloud Pak for Data .

Audit events are available for the Speech services

The IBM Cloud Pak for Data Audit Logging Service generates and forwards audit events for both the Speech to Text and Text to Speech services. The audit events match those that are available for Activity Tracker with the public service. For more information, see Audit events.

You cannot uninstall individual Speech service components

The documentation now notes that you cannot uninstall individual service components (microservices) once they are installed. To remove any of the following components, you must uninstall the Watson Speech services in their entirety and reinstall only the components that you need: Speech to Text runtime, Speech to Text asynchronous HTTP, Speech to Text customization, Text to Speech runtime, and Text to Speech customization. For more information about installing the Speech services, see Watson Speech services on IBM Cloud Pak for Data .

New French Canadian next-generation multimedia model

The service now offers a next-generation multimedia model for French Canadian: `fr-CA_Multimedia`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Updates to English next-generation telephony models

The English next-generation telephony models have been updated for improved speech recognition:

- `en-AU_Telephony`
- `en-GB_Telephony`
- `en-IN_Telephony`
- `en-US_Telephony`

All of these models continue to support low latency. You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

Italian next-generation multimedia model now supports low latency

The Italian next-generation multimedia model, `it-IT_Multimedia`, now supports low latency. For more information about next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Troubleshooting upgrade from version 4.0.x to version 4.5.x

When you upgrade the Speech services from version 4.0.x to version 4.5.x, you might encounter an issue where the PostgreSQL pods become stuck in the `Terminating` state. If this problem occurs during your upgrade, perform the following steps to resolve the problem. The information and steps are also documented in *Upgrading Watson Speech services from Version 4.0 to Version 4.5* in the *Upgrading* topic of Watson Speech services on IBM Cloud Pak for Data.

1. Use the following command to identify pods that remain in the `Terminating` state:

```
oc get pods -n ${PROJECT_CPD_INSTANCE} -o wide | awk {'print $1'}
```

1. Use the following command to set the environment variable `pods` to include the list of pods that remain in the `Terminating` state:

```
pods=$(oc get pods -n ${PROJECT_CPD_INSTANCE} -o wide | awk {'print $1'})
```

1. Use the following command to delete the stuck pods so that the upgrade process can continue:

```
pods=$(oc get pods -n ${PROJECT_CPD_INSTANCE} -o wide | grep Terminating | awk {'print $1'})
```

Defect fix: Fix custom resource entries documentation

Defect fix: The documentation for the Speech services custom resource now includes colons after the names of the models `koKrTelephony` and `nlNlTelephony`. Previously, the documentation for these two entries omitted the colons.

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a buffer over-read flaw in Linux Kernel (CVE-2020-28915)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security bypass in GNU Gzip (CVE-2022-1271)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to elevated privileges in Apple macOS Monterey and macOS Big Sur (CVE-2022-26691)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to elevated privileges in Linux Kernel (CVE-2022-27666)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to cross-site scripting in Apache Tomcat (CVE-2022-34305)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in GNU C Library (CVE-2019-19126)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in GNU C Library (CVE-2020-10029)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in GNU glibc (CVE-2020-1751)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in GNU glibc (CVE-2020-1752)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to information disclosure or denial of service in GNU glibc (CVE-2021-35942)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to buffer overflow in OpenSSL (CVE-2021-3711)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to information disclosure or denial of service in OpenSSL (CVE-2021-3712)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to weakened security in OpenSSL (CVE-2021-4160)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in OpenSSL (CVE-2022-0778)

19 August 2022 (Version 4.5.1)

Important: Deprecation date for most previous-generation models is now 3 March 2023

Superseded: This deprecation notice is superseded by the 23 February 2023 service update. The end of service date for *all* previous-generation models is now **31 July 2023**.

On 15 March 2022, the previous-generation models for all languages other than Arabic and Japanese were deprecated. At that time, the deprecated models were to remain available until 15 September 2022. To allow users more time to migrate to the appropriate next-generation models, the deprecated models will now remain available until 3 March 2023. As with the initial deprecation notice, the Arabic and Japanese previous-generation models are *not* deprecated. For complete list of all deprecated models, see the 15 March 2022 (Version 4.0.6) service update.

On 3 March 2023, the deprecated models will be removed from the service and the documentation. If you use any of the deprecated models, you must migrate to the equivalent next-generation model by the 3 March 2023.

- For more information about the next-generation models to which you can migrate from each of the deprecated models, see Previous-generation languages and models
- For more information about the next-generation models, see Next-generation languages and models
- For more information about migrating from previous-generation to next-generation models, see Migrating to next-generation models.

Note: When the previous-generation `en-US_BroadbandModel` is removed from service, the next-generation `en-US_Multimedia` model will become the default model for speech recognition requests.

3 August 2022 (Version 4.5.1)

Version 4.5.1 is now available

Speech to Text for IBM Cloud Pak for Data version 4.5.1 is now available. This version supports IBM Cloud Pak for Data version 4.5.x and Red Hat OpenShift versions 4.6, 4.8, and 4.10. For more information, see Watson Speech services on IBM Cloud Pak for Data.

Support for FIPS-enabled clusters

Both Speech to Text for IBM Cloud Pak for Data and Text to Speech for IBM Cloud Pak for Data now support running on Federal Information Processing Standard (FIPS)-enabled clusters. For more information, see Services that support FIPS.

Defect fix: Fix ephemeral storage calculations to prevent occasional pod evictions

Defect fix: A defect was fixed and calculation of ephemeral storage limits is now more precise for the Speech to Text for IBM Cloud Pak for Data and Text to Speech for IBM Cloud Pak for Data runtimes. These changes prevent occasional pod evictions when the services' runtimes are under heavy load.

Defect fix: Update speech hesitations and hesitation markers documentation

Defect fix: Documentation for speech hesitations and hesitation markers has been updated. Previous-generation models include hesitation markers in place of speech hesitations in transcription results for most languages; smart formatting removes hesitation markers from US English final transcripts. Next-generation models include the actual speech hesitations in transcription results; smart formatting has no effect on their inclusion in final transcription results.

For more information, see:

- Speech hesitations and hesitation markers
- What results does smart formatting affect?

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in rsyslog (CVE-2022-24903)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to an HTTP request smuggling issue in Twisted (CVE-2022-24801)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service, caused by a buffer overflow in Twisted (CVE-2022-21716)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service, caused by incomplete string comparison in NumPy (CVE-2021-34141)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service, caused by a buffer overflow in NumPy (CVE-2021-41496)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to cookie and authorization header exposure

in Twisted (CVE-2022-21712)

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in Perl (CVE-2018-18311)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in Perl (CVE-2018-18312)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in Perl (CVE-2018-18313)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in Perl (CVE-2018-18314)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in Perl (CVE-2018-6913)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to CRLF injection in Python (CVE-2019-11236)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in GNU Tar (CVE-2019-9923)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in Perl (CVE-2020-10543)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to an integer overflow in Perl (CVE-2020-10878)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a buffer overflow in Perl (CVE-2020-12723)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in urllib3 (CVE-2021-33503)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to injection attacks in Ansible (CVE-2021-3583)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-23772)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to incorrect access control in Golang Go (CVE-2022-23773)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-23806)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-24675)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-24921)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Golang Go (CVE-2022-28327)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a heap-based buffer overflow in libssh, caused by improper bounds checking (CVE-2021-3634)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in Python (CVE-2021-3737)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a possible sensitive information exposure in Python (CVE-2021-4189)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in lxml (CVE-2021-43818)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in MS Visual Studio (CVE-2021-21300)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a security restrictions bypass in Git (CVE-2021-40330)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution in MS Visual Studio (CVE-2022-24765)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary command execution in Git (CVE-2018-1000021)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to cross-site scripting in jQuery (CVE-2015-9251)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to cross-site scripting in jQuery (CVE-2019-

11358)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to cross-site scripting in jQuery (CVE-2020-11022)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to cross-site scripting in jQuery (CVE-2020-11023)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a data binding rules security weakness in Spring Framework (CVE-2022-22968)

29 June 2022 (Version 4.5.0)

Version 4.5.0 is now available

Speech to Text for IBM Cloud Pak for Data version 4.5.0 is now available. This version supports IBM Cloud Pak for Data version 4.5.x and Red Hat OpenShift versions 4.6, 4.8, and 4.10. For more information, see Watson Speech services on IBM Cloud Pak for Data .

Unified Speech services for IBM Cloud Pak for Data documentation

The installation and administration documentation for both Speech to Text and Text to Speech is now combined in the IBM Cloud Pak for Data documentation. For more information about installing and managing the Speech services, see Watson Speech services on IBM Cloud Pak for Data .

Changes to Speech services custom resource

The custom resource is now created when you initially install the Speech services. The process is described in the IBM Cloud Pak for Data installation documentation. The content of the custom resource has changed:

- The recommended name of the custom resource has changed from `speech-prod-cr` to `speech-cr`.
- All references to storage class have changed from variants of `storageClass` to `blockStorageClass`.
- The name of the Portworx block storage class has changed from `portworx-shared-gp3` to `portworx-db-gp3-sc`.
- The `createSecret` property has been removed for the MinIO and PostgreSQl datastores. The property is only used internally. The Speech services always use a secrets object if you create one, and they always automatically create the object if none is provided.

User-provided secrets object now supported for RabbitMQ datastore

You can now provide security credentials for the RabbitMQ datastore, just as you can for the MinIO and PostgreSQL datastores. The documented process is similar for all three datastores.

New Italian `it-IT_Multimedia` next-generation model

The service now offers a next-generation multimedia model for Italian: `it-IT_Multimedia`. The new model is generally available. It does not support low latency, but it does support language model customization and grammars. For more information about all available next-generation models, see Next-generation languages and models.

Updated Korean telephony and multimedia next-generation models

The existing Korean next-generation models have been updated:

- The `ko-KR_Telephony` model has been updated for improved low-latency support for speech recognition.
- The `ko-KR_Multimedia` model has been updated for improved speech recognition. The model now also supports low latency.

Both models are generally available, and both support language model customization and grammars. You do not need to upgrade custom language models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

Updates to multiple next-generation telephony models

The following next-generation English language telephony models have been updated for improved speech recognition:

- `en-AU_Telephony`
- `en-GB_Telephony`
- `en-IN_Telephony`
- `en-US_Telephony`

You do not need to upgrade custom models that are based on these models. For more information about all available next-generation models, see Next-generation languages and models.

Defect fix: Confidence scores are now reported for all transcription results

Defect fix: Confidence scores are now reported for all transcription results. Previously, when the service returned multiple transcripts for a single speech recognition request, confidence scores might not be returned for all transcripts.

Security vulnerabilities addressed

No security vulnerabilities were fixed for version 4.5.0.

25 May 2022 (Version 4.0.9)

Version 4.0.9 is now available

Speech to Text for IBM Cloud Pak for Data version 4.0.9 is now available. This version supports IBM Cloud Pak for Data version 4.x and Red Hat OpenShift versions 4.6 and 4.8. For more information about installing and managing the service, see Installing Watson Speech to Text.

New Brazilian Portuguese `pt-BR_Multimedia` next-generation model

The service now offers a next-generation multimedia model for Brazilian Portuguese: `pt-BR_Multimedia`. The new model supports low latency and is generally available. It also supports language model customization and grammars. For more information about the next-generation models and low latency, see

- Next-generation languages and models
- Customization support for next-generation models
- Low latency

Update to German `de-DE_Multimedia` next-generation model to support low latency

The next-generation German model, `de-DE_Multimedia`, now supports low latency. You do not need to upgrade custom models that are based on the updated German base model. For more information about the next-generation models and low latency, see

- Next-generation languages and models
- Low latency

New beta `character_insertion_bias` parameter for next-generation models

All next-generation models now support a new beta parameter, `character_insertion_bias`, which is available with all speech recognition interfaces. By default, the service is optimized for each individual model to balance its recognition of candidate strings of different lengths. The model-specific bias is equivalent to 0.0. Each model's default bias is sufficient for most speech recognition requests.

However, certain use cases might benefit from favoring hypotheses with shorter or longer strings of characters. The parameter accepts values between -1.0 and 1.0 that represent a change from a model's default. Negative values instruct the service to favor shorter strings of characters. Positive values direct the service to favor longer strings of characters. For more information, see Character insertion bias.

The Speech services do not support the OADP backup and restore utility

Watson Speech services do not support the IBM Cloud Pak for Data OpenShift APIs for Data Protection (OADP) backup and restore utility. If the Speech services are installed on a cluster, you might not be able to use the IBM Cloud Pak for Data OADP backup and restore utility to back up other services that are installed on that cluster. This limitation applies to version 4.0.0 and later versions of the Speech services.

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable a denial of service, caused by a buffer overflow with Twisted (CVE-2022-21716)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service in NumPy. (CVE-2021-33430)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a denial of service, caused by improper input validation with Spring Framework (CVE-2022-22950)

1 May 2022 (Version 1.2.x)

Important: End of service for Speech to Text version 1.2.x on IBM Cloud Pak for Data version 3.5

Important: Speech to Text version 1.2.x on IBM Cloud Pak for Data version 3.5 is out of service as of 1 May 2022. Speech to Text version 1.2.x is no longer supported, available, or documented. For more information about End of Service for Speech to Text, which is part of the Watson API Kit, see Software support discontinuance: IBM Watson API Kit for IBM Cloud Pak for Data 1.2.x .

27 April 2022 (Version 4.0.8)

Version 4.0.8 is now available

Speech to Text for IBM Cloud Pak for Data version 4.0.8 is now available. This version supports IBM Cloud Pak for Data version 4.x and Red Hat OpenShift versions 4.6 and 4.8. For more information about installing and managing the service, see Installing Watson Speech to Text.

New environment variables used in IBM Cloud Pak for Data documentation

Most commands in the Speech to Text for IBM Cloud Pak for Data documentation have been updated to use a common set of environment variables. The documentation provides a script to automatically export the environment variables before you run installation, upgrade, and administration commands. After you source the script, you can copy most commands from the documentation and run them without making any changes.

The environment variables that the script defines include the following:

- `${PROJECT_CPD_INSTANCE}` identifies the project where you plan to install IBM Cloud Pak for Data and the Speech services.
- `${PROJECT_CPD_OPS}` identifies the project for the IBM Cloud Pak for Data platform operator.
- `${PROJECT_CPFS_OPS}` identifies the project for the IBM Cloud Pak for Data foundational services.

For more information about using the environment variables, see Best practice: Setting up install variables .

The `ttsVoiceMarginalCPU` property is no longer documented

The `ttsVoiceMarginalCPU` property has been removed from the documentation for the Speech services custom resource. The property manages the tradeoff between concurrency and speech synthesis speed. The default value of `400` ensures a reasonable balance for most customers and maintains real-time synthesis.

New German next-generation multimedia model

The service now offers a next-generation multimedia model for German: `de-DE_Multimedia`. The new model is generally available. It does not support low latency. It does support language model customization and grammars as generally available functionality.

For more information about all available next-generation models and their customization support, see

- Next-generation languages and models
- Customization support for next-generation models

Beta next-generation `en-WW_Medical_Telephony` model now supports low latency

The beta next-generation `en-WW_Medical_Telephony` model now supports low latency. For more information about all next-generation models and low latency, see

- Next-generation languages and models
- Low latency

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Security Bulletin: A vulnerability with Guava affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2020-8908)
- Security Bulletin: A Google Guava vulnerability affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2018-10237)
- Security Bulletin: Vulnerabilities in Apache Tomcat affect IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2022-23181)
- Security Bulletin: A Cyrus SASL vulnerability affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2022-24407)
- Security Bulletin: A vulnerability with GNU wget affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2016-4971)
- Security Bulletin: A vulnerability with GNU Wget affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2018-0494)
- Security Bulletin: A vulnerability in 'GNU Wget' affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2018-20483)

- Security Bulletin: A vulnerability in ISC BIND affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2018-5741)
- Security Bulletin: A vulnerability in Python affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2019-20916)
- Security Bulletin: A vulnerability with ISC BIND affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2021-25214)
- Security Bulletin: A vulnerability in ISC BIND affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2021-25215)
- Security Bulletin: A vulnerability in ISC BIND affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2021-25216)
- Security Bulletin: A vulnerability in ISC BIND affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2021-25219)
- Security Bulletin: A vulnerability in PostgreSQL JDBC Driver (PgJDBC) affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2022-21724)
- Security Bulletin: A vulnerability in GNU Tar affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2019-9923)
- Security Bulletin: A vulnerability in logback-classic affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2021-42550)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a stack-based buffer overflow in GNU C Library (CVE-2022-23218)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to stack-based buffer overflow in GNU C Library (CVE-2022-23219)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to a buffer overflow and underflow in GNU C Library (CVE-2021-3999)

8 April 2022 (Version 4.0.7)

Support for sounds-like is now documented for custom models based on next-generation models

For custom language models that are based on next-generation models, support is now documented for sounds-like specifications for custom words. Support for sounds-likes has been available since late 2021.

Differences exist between the use of the `sounds_like` field for custom models that are based on next-generation and previous-generation models. For more information about using the `sounds_like` field with custom models that are based on next-generation models, see Working with custom words for next-generation models.

Important: Deprecated `customization_id` parameter removed from the documentation

Important: On 9 October 2018, the `customization_id` parameter of all speech recognition requests was deprecated and replaced by the `language_customization_id` parameter. The `customization_id` parameter has now been removed from the documentation for the speech recognition methods:

- `/v1/recognize` for WebSocket requests
- `POST /v1/recognize` for synchronous HTTP requests (including multipart requests)
- `POST /v1/recognitions` for asynchronous HTTP requests

Note: If you use the Watson SDKs, make sure that you have updated any application code to use the `language_customization_id` parameter instead of the `customization_id` parameter. The `customization_id` parameter will no longer be available from the equivalent methods of the SDKs as of their next major release. For more information about the speech recognition methods, see the API & SDK reference.

30 March 2022 (Version 4.0.7)

Version 4.0.7 is now available

Speech to Text for IBM Cloud Pak for Data version 4.0.7 is now available. This version supports IBM Cloud Pak for Data version 4.x and Red Hat OpenShift versions 4.6 and 4.8. For more information about installing and managing the service, see Installing Watson Speech to Text.

Custom resource property for specifying a default model

The default voice for speech recognition requests is `en-US_BroadbandModel`. If you do not install the `en-US_BroadbandModel`, you must either

- Use the `model` parameter to pass the voice that is to be used with each request.
- Specify a new default model for your installation of Speech to Text for IBM Cloud Pak for Data by using the `defaultSTTModel` property in the Speech services custom resource. For more information, see Installing Watson Speech to Text and Using the default model.

Updates to English and French next-generation multimedia models to support low latency

The following multimedia models have been updated to support low latency:

- Australian English: `en-AU_Multimedia`
- UK English: `en-GB_Multimedia`
- US English: `en-US_Multimedia`
- French: `fr-FR_Multimedia`

You do not need to upgrade custom language models that are built on these base models. For more information about the next-generation models and low latency, see

- Next-generation languages and models
- Low latency

New Castilian Spanish next-generation multimedia model

The service now offers a next-generation multimedia model for Castilian Spanish: `es-ES_Multimedia`. The new model supports low latency and is generally available. It also supports language model customization and grammars.

For more information about all available next-generation models and their customization support, see

- Next-generation languages and models
- Customization support for next-generation models

Beta next-generation `en-WW_Medical_Telephony` model now supports smart formatting

The beta next-generation `en-WW_Medical_Telephony` model now supports the `smart_formatting` parameter for US English audio. For more information about all next-generation models, see Next-generation languages and models

Security vulnerabilities addressed

The following security vulnerabilities have been fixed:

- Red Hat CVE-2022-24407: A flaw was found in the SQL plugin shipped with Cyrus SASL. The vulnerability occurs due to failure to properly escape SQL input and leads to an improper input validation vulnerability. This flaw allows an attacker to execute arbitrary SQL commands and the ability to change the passwords for other accounts allowing escalation of privileges.
- Security Bulletin: A jwt-go vulnerability affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2020-26160)
- Security Bulletin: A vulnerability in Golang Go affects IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2021-29923)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is affected but not classified as vulnerable by a remote code execution in Spring Framework (CVE-2022-22965)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to arbitrary code execution with IBM WebSphere Application Server (CVE-2021-23450)

17 March 2022 (Version 4.0.6)

Grammar support for next-generation models is now generally available

Grammar support is now generally available (GA) for next-general models that meet the following conditions:

- The models are generally available.
- The models support language model customization.

For more information, see the following topics:

- For more information about the status of grammar support for next-generation models, see Customization support for next-generation models.
- For more information about grammars, see Grammars.

15 March 2022 (Version 4.0.6)

Important: Deprecation of most previous-generation models

Superseded: This deprecation notice is superseded by the [23 February 2023 service update](). The end of service date for *all* previous-generation models is now **31 July 2023**.

Effective 15 March 2022, previous-generation models for all languages other than Arabic and Japanese are deprecated. The deprecated models remain available until 15 September 2022, when they will be removed from the service and the documentation. The Arabic and Japanese previous-generation models are *not* deprecated.

The following previous-generation models are now deprecated:

- Chinese (Mandarin): `zh-CN_NarrowbandModel` and `zh-CN_BroadbandModel`
- Dutch (Netherlands): `nl-NL_NarrowbandModel` and `nl-NL_BroadbandModel`
- English (Australian): `en-AU_NarrowbandModel` and `en-AU_BroadbandModel`
- English (United Kingdom): `en-UK_NarrowbandModel` and `en-UK_BroadbandModel`
- English (United States): `en-US_NarrowbandModel`, `en-US_BroadbandModel`, and `en-US_ShortForm_NarrowbandModel`
- French (Canadian): `fr-CA_NarrowbandModel` and `fr-CA_BroadbandModel`
- French (France): `fr-FR_NarrowbandModel` and `fr-FR_BroadbandModel`
- German: `de-DE_NarrowbandModel` and `de-DE_BroadbandModel`
- Italian: `it-IT_NarrowbandModel` and `it_IT_BroadbandModel`
- Korean: `ko-KR_NarrowbandModel` and `ko-KR_BroadbandModel`
- Portuguese (Brazilian): `pt-BR_NarrowbandModel` and `pt-BR_BroadbandModel`
- Spanish (Argentinian): `es-AR_NarrowbandModel` and `es-AR_BroadbandModel`
- Spanish (Castilian): `es-ES_NarrowbandModel` and `es-ES_BroadbandModel`
- Spanish (Chilean): `es-CL_NarrowbandModel` and `es-CL_BroadbandModel`
- Spanish (Colombian): `es-CO_NarrowbandModel` and `es-CO_BroadbandModel`
- Spanish (Mexican): `es-MX_NarrowbandModel` and `es-MX_BroadbandModel`
- Spanish (Peruvian): `es-PE_NarrowbandModel` and `es-PE_BroadbandModel`

If you use any of these deprecated models, you must migrate to the equivalent next-generation model by the end of service date.

- For more information about the next-generation models to which you can migrate from each of the deprecated models, see [Previous-generation languages and models]()
- For more information about the next-generation models, see [Next-generation languages and models]()
- For more information about migrating from previous-generation to next-generation models, see [Migrating to next-generation models]().

Note: When the previous-generation `en-US_BroadbandModel` is removed from service on 15 September, the next-generation `en-US_Multimedia` model will become the default model for speech recognition requests.

Next-generation models now support audio-parsing parameters

All next-generation models now support the following audio-parsing parameters as generally available features:

- `end_of_phrase_silence_time` specifies the duration of the pause interval at which the service splits a transcript into multiple final results. For more information, see [End of phrase silence time]().
- `split_transcript_at_phrase_end` directs the service to split the transcript into multiple final results based on semantic features of the input. For more information, see [Split transcript at phrase end]().

Defect fix: Correct speaker labels documentation

Defect fix: Documentation of speaker labels included the following erroneous statement in multiple places: *For next-generation models, speaker labels are not supported for use with interim results or low latency.* Speaker labels are supported for use with interim results and low latency for next-generation models. For more information, see [Speaker labels]().

23 February 2022 (Version 4.0.6)

Version 4.0.6 is now available

Speech to Text for IBM Cloud Pak for Data version 4.0.6 is now available. This version supports IBM Cloud Pak for Data version 4.x and Red Hat

OpenShift versions 4.6 and 4.8. For more information about installing and managing the service, see Installing Watson Speech to Text.

Updates to import/export scripts

The `import_export.sh` and `transfer_ownership.sh` scripts have been updated. These scripts are used to import and export data between clusters, back up and restore data, and migrate data from version 3.5 to version 4.0.x. The scripts have been modified and improved as follows:

- The `transfer_ownership.sh` script now requires a `-c` option to be included on the command line before the `<custom_resource_name>` argument.
- The `transfer_ownership.sh` script now requires a `-v <version>` option and argument to indicate the version to which ownership of resources is being transferred. Specify `35` for version 3.5 or `40` for version 4.0.x.
- The `transfer_ownership.sh` script now requires a `-p` option to be included on the command line before the `<postgres_auth_secret_name>` argument.
- The `<postgres_auth_secret_name>` argument provides the Kubernetes secret that is used to authenticate to the PostgreSQL datastore to which you are transferring ownership. You can omit the authentication secret if is the same as the default value (`<custom-resource-name>-postgres-auth-secret` for version 4.0.x, `user-provided-postgressql` for version 3.5). You must provide the secret if it is different from the default value.
- Both scripts now include a `-h` (`--help`) option to display information about the script and its usage.

For more information, see

- Administering Watson Speech to Text, specifically *Importing and exporting data* and *Backing up and restoring data* .
- Upgrading Watson Speech to Text, specifically *Migrating data from IBM Cloud Pak for Data Version 3.5* .

Updated recommendation for OpenShift Container Storage

Starting with Speech services version 4.0.6, the recommended storage class for OpenShift Container Storage is `ocs-storagecluster-ceph-rbd`.

- If you are installing Speech services 4.0.6 or upgrading to Speech services 4.0.6 from IBM Cloud Pak for Data version 3.5, specify the `ocs-storagecluster-ceph-rbd` storage class during installation or upgrade.
- If you are upgrading to Speech services 4.0.6 from a previous refresh of Cloud Pak for Data version 4.0, continue to use `ocs-storagecluster-cephfs`. You cannot change the storage that is used in an existing deployment.

The value is specified with the `storageClass` property in the Speech services custom resource:

```
################
# Storage class
################
  storageClass: "ocs-storagecluster-ceph-rbd"
```

The Speech services work with either version of OpenShift Container Storage. The newly recommended version has more restrictive access permissions. For more information, see

- Installing Watson Speech to Text
- Upgrading Watson Speech to Text

New beta `en-WW_Medical_Telephony` model is now available

A new beta next-generation `en-WW_Medical_Telephony` is now available. The new model understands terms from the medical and pharmacological domains. Use the model in situations where you need to transcribe common medical terminology such as medicine names, product brands, medical procedures, illnesses, types of doctor, or COVID-19-related terminology. Common use cases include conversations between a patient and a medical provider (for example, a doctor, nurse, or pharmacist).

The new model is installed from the Speech services custom resource by setting `enWwMedicalTelephony` to `enabled: true`. The model is available for all supported English dialects: Australian, Indian, UK, and US.

- The model supports language model customization and grammars as beta functionality.
- It supports most of the same parameters as the `en-US_Telephony` model.
- It does *not* support the following parameters: `low_latency`, `profanity_filter`, `redaction`, and `speaker_labels`.
- At this time, it does *not* support `smart_formatting` for IBM Cloud Pak for Data.

For more information, see The English medical telephony model.

Update to Chinese `zh-CN_Telephony` model

The next-generation Chinese model `zh-CN_Telephony` has been updated for improved speech recognition. The model continues to support low latency. By default, the service automatically uses the updated model for all speech recognition requests. For more information about all available next-generation models, see Next-generation languages and models.

If you have custom language models that are based on the updated model, you must upgrade your existing custom models to take advantage of the updates by using the `POST /v1/customizations/{customization_id}/upgrade_model` method. For more information, see Upgrading custom models.

Update to Japanese `ja-JP_Multimedia` model to support low latency

The next-generation Japanese model `ja-JP_Multimedia` now supports low latency. You can use the `low_latency` parameter with speech recognition requests that use the model. You do not need to upgrade custom models that are based on the updated Japanese base model. For more information about the next-generation models and low latency, see Next-generation languages and models and Low latency.

11 February 2022 (Version 4.0.5)

Defect fix: Improve custom model upgrade and base model version documentation

Defect fix: The documentation that describes the upgrade of custom models and the version strings that are used for different versions of base models has been updated. The documentation now states that upgrade for language model customization also applies to next-generation models. Also, the version strings that represent different versions of base models have been updated. And the `base_model_version` parameter can also be used with upgraded next-generation models.

For more information about custom model upgrade, when upgrade is necessary, and how to use older versions of custom models, see

- Upgrading custom models
- Using upgraded custom models for speech recognition

Defect fix: Update capitalization documentation

Defect fix: The documentation that describes the service's automatic capitalization of transcripts has been updated. The service capitalizes appropriate nouns only for the following languages and models:

- All previous-generation US English models
- The next-generation German model

For more information, see Capitalization.

31 January 2022 (Version 4.0.5)

Version 4.0.5 has been updated

Speech to Text for IBM Cloud Pak for Data version 4.0.5 has been updated to address installation issues. The case package version is now 4.0.6. Use this package instead of the version 4.0.5 package. For more information about installing and managing the service, see Installing Watson Speech to Text.

Important: Extra steps for mirrored installation are no longer necessary

Important: The 26 January 2022 release notes included important notes for the following steps:

- Additional step for performing a mirrored installation of Minio datastore
- Additional steps for performing a mirrored installation of new next-generation models

These additional steps are no longer needed. The case package has been updated to correct the installation issues.

26 January 2022 (Version 4.0.5)

Version 4.0.5 is now available

Speech to Text for IBM Cloud Pak for Data version 4.0.5 is now available. This version supports IBM Cloud Pak for Data version 4.x and Red Hat

OpenShift versions 4.6 and 4.8. For more information about installing and managing the service, see Installing Watson Speech to Text.

Important: Additional step for performing a mirrored installation of Minio datastore

Important: These steps are no longer needed if you install case package 4.0.6. For more information, see 31 January 2022 (Version 4.0.5) .

If you are performing a mirrored installation (for example, in an air-gapped environment), you need to perform an additional step *before* completing either of the following steps:

- Step 7 Mirroring the images to the private registry of *Mirroring images with a bastion model*
- Step 8 Mirroring the images to the intermediary container registry of *Mirroring images with an intermediary container registry*

This step is mandatory to copy the necessary images for the Minio datastore:

```
echo 'cp.icr.io,cp/opencontent-minio-
client,1.1.4,sha256:7b4cf5e47a0455cfa7ca9ab246b80916e4dccbc1483b3e0f276fb7b0ab3e5c60,IMAGE,linux,x86_64,"",0,CASE,"",""' \
>> $CASE_PATH/ibm-watson-speech-4.0.5-images.csv
```

Failure to perform this step will cause installation errors for both Speech to Text and Text to Speech.

Important: Additional steps for performing a mirrored installation of new next-generation models

Important: These steps are no longer needed if you install case package 4.0.6. For more information, see 31 January 2022 (Version 4.0.5) .

If you are performing a mirrored installation (for example, for an air-gapped environment) and plan to install any of the new next-generation models for Speech to Text (for more information, see the later release note), you must perform an additional step *before* completing either of the following steps:

- Step 7 Mirroring the images to the private container registry of *Mirroring images with a bastion model*
- Step 8 Mirroring the images to the intermediary container registry of *Mirroring images with an intermediary container registry*

Each additional step is unique to the model that is being installed. If you install more than one of the new models, issue the indicated command for each model that you are installing.

- For the Chinese telephony model (`zh-CN_Telephony`):

  ```
  echo 'cp.icr.io,cp/watson-speech/zh-cn-telephony,2022-01-05-
  405models,sha256:52af6dfccd64ccd81b409936442a51a71f4ee96d980e1fc6a343a05bd4ed7fbc,IMAGE,linux,x86_64,"",0,CASE,"",""' \
  >> $CASE_PATH/ibm-watson-speech-4.0.5-images.csv
  ```

- For the Latin American Spanish telephony model (`es-LA_Telephony`):

  ```
  echo 'cp.icr.io,cp/watson-speech/es-la-telephony,2022-01-05-
  405models,sha256:58e8c04abe9659472e89bf0778b7dc66e0ddceb4ea18d9d3e048a08c72125ea2,IMAGE,linux,x86_64,"",0,CASE,"",""' \
  >> $CASE_PATH/ibm-watson-speech-4.0.5-images.csv
  ```

- For the Australian English multimedia model (`en-AU_Multimedia`):

  ```
  echo 'cp.icr.io,cp/watson-speech/en-au-multimedia,2022-01-05-
  405models,sha256:167f9a76258530a56a6abdd1c311f2ea05d6820ee0e802fbf2f96f08fb8a7646,IMAGE,linux,x86_64,"",0,CASE,"",""' \
  >> $CASE_PATH/ibm-watson-speech-4.0.5-images.csv
  ```

- For the UK English multimedia model (`en-GB_Multimedia`):

  ```
  echo 'cp.icr.io,cp/watson-speech/en-gb-multimedia,2022-01-05-
  405models,sha256:167f9a76258530a56a6abdd1c311f2ea05d6820ee0e802fbf2f96f08fb8a7646,IMAGE,linux,x86_64,"",0,CASE,"",""' \
  >> $CASE_PATH/ibm-watson-speech-4.0.5-images.csv
  ```

License Server is now automatically installed

The Speech services operator now automatically installs the required License Server when it installs the Speech services. You no longer need to install the License Server from the IBM Cloud Pak for Data foundational services, and you no longer need to use additional YAML content to create an OperandRequest with the necessary bindings.

Removal of steps specific to PostgreSQL EnterpriseDB server

The previous version of the documentation included steps for the PostgreSQL EnterpriseDB server that were specific to the Speech services. These steps were documented in the topics *Upgrading Watson Speech to Text (Version 4.0)* and *Uninstalling Watson Speech to Text* . These additional steps are no longer necessary and have been removed from the documentation.

RabbitMQ datastore is now used only by the `sttAsync` component

The RabbitMQ datastore was previously used by components of both Speech services, Speech to Text and Text to Speech. It now handles non-persistent message queuing for the Speech to Text asynchronous HTTP component (`sttAsync`) only. It is used only if the `sttAsync` component is installed and enabled.

New next-generation models

The service now supports the following next-generation models with Speech to Text for IBM Cloud Pak for Data:

- Chinese (Mandarin) telephony model (`zh-CN_Telephony`). The new model supports low latency.
- English (Australian) multimedia model (`en-AU_Multimedia`). The new model does not support low latency.
- English (UK) multimedia model (`en-GB_Multimedia`). The new model does not support low latency.
- Spanish (Latin American) telephony model (`es-LA_Telephony`). The new model supports low latency.

Note: The Latin American Spanish model, `es-LA_Telephony`, applies to all Latin American dialects. It is the equivalent of the previous-generation models that are available for the Argentinian, Chilean, Colombian, Mexican, and Peruvian dialects. If you used a previous-generation model for any of these specific dialects, use the `es-LA_Telephony` model to migrate to the equivalent next-generation model.

The new models are generally available for speech recognition. They are generally available for language model customization and beta for grammars. They are not supported for acoustic model customization.

- **Important:** If you are performing a mirrored installation (for example, in an air-gapped environment) and plan to install any of the new next-generation models for Speech to Text, you must perform additional steps *before* mirroring the images. For more information, see the earlier release note.
- For more information about using the custom resource to install models, see Installing Watson Speech to Text.
- For more information about all available next-generation models, see Next-generation languages and models.
- For more information about customization support for next-generation models, see Customization support for next-generation models.

Next-generation US English models are now installed by default

The next-generation US English models, `en-US_Multimedia` and `en-US_Telephony`, are now installed by default with Speech to Text for IBM Cloud Pak for Data. These models join `en-US_BroadbandModel`, `en-US_NarrowbandModel`, `en-US_ShortForm_NarrowbandModel` as the models that are installed by default. The models now have the following entries in the Speech services custom resource:

```
########################################
# Speech to Text next-generation models
########################################
      enUsMultimedia:    # US English (en-US) Multimedia model
        enabled: true
      enUsTelephony:     # US English (en-US) Telephony model
        enabled: true
```

For more information about using the custom resource to install models, see Installing Watson Speech to Text.

Security vulnerabilities addressed

The following security vulnerabilities associated with Apache Log4j have been fixed:

- Security Bulletin: Vulnerability in Apache Log4j may affect IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2021-4104)
- Security Bulletin: IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data is vulnerable to denial of service and arbitrary code execution due to Apache Log4j (CVE-2021-45105 and CVE-2021-45046)

20 December 2021 (Version 4.0.4)

Version 4.0.4 is now available

Speech to Text for IBM Cloud Pak for Data version 4.0.4 is now available. This version supports IBM Cloud Pak for Data version 4.x and Red Hat OpenShift versions 4.6 and 4.8. For more information about installing and managing the service, see Installing Watson Speech to Text.

Important: Changes to properties for disabling the storage and logging of user data

Important: The names of the properties of the Speech services custom resource that specify whether user data is stored and logged have changed. The custom resource formerly contained the following properties:

```
#################
# Anonymize logs
#################
  sttRuntime:
    anonymizeLogs: "false"  # If true, disables storage and logging of user data
  sttAMPatcher:
    anonymizeLogs: "false"  # If true, disables storage and logging of user data
  ttsRuntime:
    anonymizeLogs: "false"  # If true, disables storage and logging of user data
```

These properties are now named as follows:

```
###################################
# Storage and logging of user data
###################################
  sttRuntime:
    skipAudioAndResultLogging: "false"  # If true, disables storage and logging of user data
  sttAMPatcher:
    skipAudioAndResultLogging: "false"  # If true, disables storage and logging of user data
  ttsRuntime:
    skipAudioAndResultLogging: "false"  # If true, disables storage and logging of user data
```

If you already set these properties in your custom resource to change the default value of `false` to `true`, you need to edit your custom resource. You must manually change the names of the properties to the new values and save the updated custom resource. For more information, see [Installing Watson Speech to Text](#).

Important: Changes to properties of PostgreSQL secrets object

Important: When you install the Speech services, an object that contains a randomly generated password for the PostgreSQL datastore is created by default. You can choose instead to specify the password manually. If you do, the properties of the YAML file for the secrets object have changed. For more information, see the topic about managing your datastores in [Administering Watson Speech to Text](#).

Important: PostgreSQL pods do not start with EnterpriseDB version 1.10 operator

Important: With Speech to Text for IBM Cloud Pak for Data version 4.0.3, PostgreSQL pods based on the EnterpriseDB version 1.10 operator can fail to start. This prevents the Speech services from starting. A workaround exists for this problem. If your Speech services fail to start, see [PostgreSQL pods do not start with EnterpriseDB version 1.10 operator](#) for information about diagnosing and resolving the problem.

This problem is fixed in Speech to Text for IBM Cloud Pak for Data version 4.0.4.

New support for IBM Spectrum Scale Container Native storage class

Since version 4.0.3, the Speech services support the IBM Spectrum® Scale Container Native storage class. To use IBM Spectrum Scale, specify `"ibm-spectrum-scale-sc"` for the `storageClass` property of the Speech services custom resource. For more information, see [Installing Watson Speech to Text](#).

Interaction of Speech services with MinIO datastore during installation

The Speech services runtime components, `sttRuntime` and `ttsRuntime`, cannot start until the models and voices for the services are fully uploaded into the MinIO datastore. During installation, the services might fail and automatically restart themselves one or more times until upload of the models and voices is complete. They then start properly. No user action is required.

Defect fix: Correct upgrade documentation

Defect fix: Documentation for upgrading the Speech services to new versions of IBM Cloud Pak for Data version 4.0.x included incorrect references in some commands. These references are now correct:

- The strings `watsonSpeechToTextStatus` and `watsonTextToSpeechStatus` have been changed to `speechStatus` in both cases.
- The strings `status.watsonSpeechToTextVersion` and `status.watsonTextToSpeechVersion` have been changed to `.spec.version` in both cases.

For more information, see [Upgrading Watson Speech to Text](#).

Important: Custom language models based on certain next-generation models must be re-created

Important: If you created custom language models based on certain next-generation models, you must re-create the custom models. Until you re-create the custom language models, speech recognition requests that attempt to use the custom models fail with HTTP error code 400.

You need to re-create custom language models that you created based on the following versions of next-generation models:

- For the `en-AU_Telephony` model, custom models that you created from `en-AU_Telephony.v2021-03-03` to `en-AU_Telephony.v2021-10-04`.
- For the `en-GB_Telephony` model, custom models that you created from `en-GB_Telephony.v2021-03-03` to `en-GB_Telephony.v2021-10-04`.
- For the `en-US_Telephony` model, custom models that you created from `en-US_Telephony.v2021-06-17` to `en-US_Telephony.v2021-10-04`.
- For the `en-US_Multimedia` model, custom models that you created from `en-US_Multimedia.v2021-03-03` to `en-US_Multimedia.v2021-10-04`.

To identify the version of a model on which a custom language model is based, use the `GET /v1/customizations` method to list all of your custom language models or the `GET /v1/customizations/{customization_id}` method to list a specific custom language model. The `versions` field of the output shows the base model for a custom language model. For more information, see Listing custom language models.

To re-create a custom language model, first create a new custom model. Then add all of the previous custom model's corpora and custom words to the new model. You can then delete the previous custom model. For more information, see Creating a custom language model.

Updates to multiple next-generation models for improved speech recognition

The following next-generation models have been updated for improved speech recognition:

- Australian English telephony model (`en-AU_Telephony`)
- UK English telephony model (`en-GB_Telephony`)
- US English multimedia model (`en-US_Multimedia`)
- US English telephony model (`en-US_Telephony`)
- Castilian Spanish telephony model (`es-ES_Telephony`)

For more information about all available next-generation models, see Next-generation languages and models.

New beta grammar support for next-generation models

Grammar support is now available as beta functionality for all available next-generation models. All next-generation models are generally available (GA) and support language model customization. For more information, see the following topics:

- For more information about the status of grammar support for next-generation models, see Customization support for next-generation models.
- For more information about grammars, see Grammars.

New `custom_acoustic_model` field for supported features

The `GET /v1/models` and `GET /v1/models/{model_id}` methods now report whether a model supports acoustic model customization. The `SupportedFeatures` object now includes an additional field, `custom_acoustic_model`, a boolean that is `true` for a model that supports acoustic model customization and `false` otherwise. Currently, the field is `true` for all previous-generation models and `false` for all next-generation models.

- For more information about these methods, see Listing information about models.
- For more information about support for acoustic model customization, see Language support for customization.

Security vulnerability addressed

The following security vulnerability associated with Apache Log4j has been fixed:

- Security Bulletin: Vulnerability in Apache Log4j may affect IBM Watson Speech Services Cartridge for IBM Cloud Pak for Data (CVE-2021-4428)

20 December 2021 (Version 1.2.x)

Important: You can no longer install Speech to Text version 1.2.x on IBM Cloud Pak for Data version 3.5

Important: You can no longer perform new installations of Speech to Text version 1.2.x on IBM Cloud Pak for Data version 3.5. You can install only Speech to Text version 4.0.x on IBM Cloud Pak for Data version 4.x. For more information, see Installing Watson Speech to Text.

The Speech services for IBM Cloud Pak for Data version 3.5 reach their End of Support date on 30 April 2022. You are encouraged to upgrade to the latest version 4.0.x release of the services at your earliest convenience. For more information, see Upgrading Watson Speech to Text.

30 November 2021 (Version 4.0.3)

Version 4.0.3 is now available

Speech to Text for IBM Cloud Pak for Data version 4.0.3 is now available. This version supports IBM Cloud Pak for Data version 4.x and Red Hat OpenShift versions 4.6 and 4.8. For more information about installing and managing the service, see Installing Watson Speech to Text.

License Server now a mandatory prerequisite

You must now install the License Server from the IBM Cloud Pak for Data foundational services. You must install the License Server by using the YAML content that is provided to create an OperandRequest with the necessary bindings. You must also install the License Service in the same namespace as the service (operand), which is also where IBM Cloud Pak for Data is installed. For more information, see Installing Watson Speech to Text.

New support for in-place upgrade

The service now supports in-place, operator-based upgrade from version 4.0.0 to version 4.0.3. Moving from IBM Cloud Pak for Data version 3.5 to version 4.0.3 continues to require use of migration utilities. For more information, see Upgrading Watson Speech to Text.

EDB PostgreSQL operator and license installation changes

Installation, upgrade, and uninstallation for the Enterprise DB PostgreSQL operator and license have changed:

- Instructions for installing the EDB PostgreSQL operator and license are now included with the IBM Cloud Pak for Data foundational services. The instructions for installing the Speech services have been updated accordingly. For more information, see Installing Watson Speech to Text.
- Instructions for upgrading from Speech to Text version 4.0.0 to 4.0.3 include instructions for uninstalling the previous EDB PostgreSQL operator and license and reinstalling them with the IBM Cloud Pak for Data foundational services. For more information, see Upgrading Watson Speech to Text.
- Instructions for uninstalling the Speech services now include steps for removing the EDB PostgreSQL operator and license that were previously installed with Speech to Text. For more information, see Uninstalling Watson Speech to Text.

New guidance for scaling up your installation

The service now provides updated guidance about scaling up your installation. The information includes specifying the number of pods, the number of CPUs allocated per pod, and the maximum number of concurrent sessions with previous- and next-generation models. For more information, see Administering Watson Speech to Text.

Command-line updates to import and export utilities

The commands that are used with the import and export utilities for the Speech services include new options and arguments. The import and export utilities are also the foundation for backing up and restoring the services and for migrating from IBM Cloud Pak for Data version 3.5 to version 4.0.3. For more information about using the utilities, see

- Administering Watson Speech to Text
- Upgrading Watson Speech to Text

New property for specifying the CPUs for acoustic model training

The `sttAMPatcher` microservice manages acoustic model customization for the service. The AM Patcher uses a dedicated number of CPUs to handle requests. You can use the new `sttAMPatcher.resources.requestsCPU` property to increase the number of CPUs that are dedicated to handling acoustic model training requests by the AM Patcher. This may be necessary if you experience training failures during acoustic model training. For more information, see Installing Watson Speech to Text.

New next-generation models

The service now supports the following new next-generation language models. All of the new models are generally available.

- Czech: `cs-CZ_Telephony`. The model supports low latency.
- Belgian Dutch (Flemish): `nl-BE_Telephony`. The model supports low latency.
- French: `fr-FR_Multimedia`. The new model does not support low latency.

- Indian English: `en-IN_Telephony`. The model supports low latency.
- Indian Hindi: `hi-IN_Telephony`. The model supports low latency.
- Japanese: `ja-JP_Multimedia`. The model does not support low latency.
- Korean: `ko-KR_Multimedia`. The model does not support low latency.
- Korean: `ko-KR_Telephony`. The model supports low latency.
- Netherlands Dutch: `nl-NL_Telephony`. The model supports low latency.

For more information about all next-generation models and about low latency, see Next-generation languages and models and Low latency.

Updates to next-generation models

The following next-generation models have been updated for improved speech recognition. All of the models are generally available.

- Arabic: `ar-MS_Telephony`. The model now supports low latency.
- Brazilian Portuguese: `pt-BR_Telephony`. The model continues to support low latency.
- US English: `en-US_Telephony`. The model continues to support low latency.
- Canadian French: `fr-CA_Telephony`. The model now supports low latency.
- Italian: `it-IT_Telephony`. The model now supports low latency.

For more information about all next-generation models and about low latency, see Next-generation languages and models and Low latency.

Defect fix: Address asynchronous HTTP failures

Defect fix: The asynchronous HTTP interface failed to transcribe some audio. In addition, the callback for the request returned status `recognitions.completed_with_results` instead of `recognitions.failed`. This error has been resolved.

Defect fix: Improve speakers labels results

Defect fix: When you use speakers labels with next-generation models, the service now identifies the speaker for all words of the input audio, including very short words that have the same start and end timestamps.

Defect fix: Update interim results and low-latency documentation

Defect fix: Documentation that describes the interim results and low-latency features with next-generation models has been rewritten for clarity and correctness. For more information, see the following topics:

- Interim results and low latency, especially Requesting interim results and low latency
- How the service sends recognition results

Defect fix: Correct multitenancy documentation

Defect fix: The IBM Cloud Pak for Data topic Multitenancy support incorrectly stated that the Speech services do not support multitenancy. The topic has been updated to state that the Speech services support the following operations:

- Install the service in separate projects
- Install the service multiple times in the same project
- Install the service once and deploy multiple instances in the same project

The documentation that is specific to the Speech services correctly stated the multitenancy support.

1 October 2021 (Version 1.1.x)

Version 1.1.x is out of service

Speech to Text and Text to Speech for IBM Cloud Pak for Data version 1.1.x went out of service on 30 September 2021. As of 1 October 2021, the documentation for version 1.1.x is no longer available. For more information, see Software withdrawal and support discontinuance.

31 August 2021 (Version 4.0.0)

All next-generation models are now generally available

All next-generation language models are now generally available (GA). They are supported for use in production environments and applications.

- For more information about all next-generation language models and which models are currently available for IBM Cloud Pak for Data, see Next-generation languages and models.
- For more information about the features that are supported for each next-generation model, see Supported features for next-generation models.

Language model customization for next-generation models is now generally available

Language model customization is now generally available (GA) for all available next-generation languages and models. Language model customization for next-generation models is supported for use in production environments and applications.

You use the same commands to create, manage, and use custom language models, corpora, and custom words for next-generation models as you do for previous-generation models. But customization for next-generation models works differently from customization for previous-generation models. For custom models that are based on next-generation models:

- The custom models have no concept of out-of-vocabulary (OOV) words.
- Words from corpora are not added to the words resource.
- You cannot currently use the sounds-like feature for custom words.
- You do not need to upgrade custom models when base language models are updated.
- Grammars are not currently supported.

For more information about using language model customization for next-generation models, see

- Understanding customization
- Language support for customization
- Creating a custom language model
- Using a custom language model for speech recognition
- Working with corpora and custom words for next-generation models

Additional topics describe managing custom language models, corpora, and custom words.

29 July 2021 (Version 4.0.0)

Version 4.0.0 is available

IBM Watson® Speech to Text for IBM Cloud Pak® for Data version 4.0.0 is now available. Installation and administration of the service include many changes. This version supports IBM Cloud Pak for Data version 4.x and Red Hat OpenShift version 4.6. For more information about installing and managing the service, see Installing IBM Watson Speech to Text for IBM Cloud Pak for Data.

New next-generation language models

The service now supports a growing number of next-generation language models. The next-generation *multimedia* and *telephony* models improve upon the speech recognition capabilities of the service's previous generation of broadband and narrowband models. The new models leverage deep neural networks and bidirectional analysis to achieve both higher throughput and greater transcription accuracy.

At this time, the next-generation language models and the `low_latency` parameter are beta functionality. The next-generation models support a limited number of languages and speech recognition features. The supported languages, models, and features will increase with future releases.

Many of the next-generation models also support a new `low_latency` parameter that lets you request faster results at the possible expense of reduced transcription quality. When low latency is enabled, the service curtails its analysis of the audio, which can reduce the accuracy of the transcription. This trade-off might be acceptable if your application requires lower response time more than it does the highest possible accuracy.

The `low_latency` parameter impacts your use of the `interim_results` parameter with the WebSocket interface. Interim results are available only for those next-generation models that support low latency, and only if both the `interim_results` and `low_latency` parameters are set to `true`.

- For more information about the next-generation models and their capabilities, see Next-generation languages and models.
- For more information about language support for next-generation models and about which next-generation models support low latency, see Supported next-generation language models.
- For more information about feature support for next-generation models, see Supported features for next-generation models.
- For more information about the `low_latency` parameter, see Low latency.
- For more information about the interaction between the `low_latency` and `interim_results` parameters for next-generation models, see

Requesting interim results and low latency.

Arabic language broadband model renamed

The Arabic language broadband model is now named `ar-MS_BroadbandModel`. The former name, `ar-AR_BroadbandModel`, is deprecated. It will continue to function for at least one year but might be removed at a future date. You are encouraged to migrate to the new name at your earliest convenience.

Unified Speech to Text documentation

The documentation for IBM Watson Speech to Text for IBM Cloud Pak for Data is now combined with the documentation for managed instances of the Speech to Text service that are hosted on IBM Cloud. This is true of both the guide and reference documentation for the two forms of the service. Links to the formerly separate version of the IBM Cloud Pak for Data documentation for the service redirect to the unified documentation.

For more information about identifying information that pertains to only one version of the product, see About Speech to Text.

Defect fix: Improve documentation

Defect fix: The documentation has been updated to correct the following information:

- The documentation failed to state that next-generation models do *not* produce hesitation markers. The documentation has been updated to note that only previous-generation models produce hesitation markers. Next-generation models include the actual hesitations in transcription results. For more information, see Speech hesitations and hesitation markers.
- The documentation incorrectly stated that using the `smart_formatting` parameter causes the service to remove hesitation markers from final transcription results for Japanese. Smart formatting does not remove hesitation markers from final results for Japanese, only for US English. For more information, see What results does smart formatting affect?

Version 1.1.x is going out of service

Speech to Text and Text to Speech for IBM Cloud Pak for Data version 1.1.x go out of service on **30 September 2021**. You must upgrade to a later version of the services on IBM Cloud Pak for Data before that date. As of 1 October 2021, the documentation for version 1.1.4 will no longer be available.

12 April 2021 (Version 1.2.1)

Addition to `speech-override.yaml` file

The minimal `speech-override.yaml` file includes an extra definition, `dockerRegistryPrefix`:

```
global:
  dockerRegistryPrefix: "{Registry}"
  image:
    pullSecret: "{Registry_pull_secret}"
```

`{Registry}` is the path for the internal Docker registry. It must be `image-registry.openshift-image-registry.svc:5000/{namespace}`, where `{namespace}` is the namespace in which IBM Cloud Pak® for Data is installed, normally `zen`.

9 April 2021 (Version 1.2.1)

Support for modifying installed models and voices

The Speech services let you add or remove installed models and voices for version 1.2 or 1.2.1 of the services.

Version 1.2.1 (26 March 2021)

Version 1.2.1 is available

Speech to Text for IBM Cloud Pak for Data version 1.2.1 is now available. Versions 1.2 and 1.2.1 use the same version 1.2 documentation and installation instructions. Version 1.2.1 supports installation on Red Hat OpenShift version 4.6 in addition to versions 4.5 and 3.11.

New installation instructions

For both clusters connected to the internet and air-gapped clusters, the installation instructions include the following steps:

- Use the `oc label` command to set up required labels for the namespace where IBM Cloud Pak for Data is installed.
- Use the `oc project` command to ensure that you are pointing at the correct OpenShift project.
- Use the `cpd-cli install` command to install an Enterprise DB PostgreSQL server that is used by the Speech services.

You perform these steps before you install the Speech services.

New uninstallation instructions

A step was added to the procedure for uninstalling the Speech services to clean up all of the resources from the installation.

Entitled registry for PostgreSQL datastore

The entitled registry path from which the service pulls images for the PostgreSQL datastore has changed. The registry location changed from `cp.icr.io/cp/watson-speech` to `cp.icr.io/cp/cpd`. This change is transparent to users.

Secrets for Minio and PostgreSQL datastores

The Minio and PostgreSQL datastores require the following hard-coded values for their secrets:

- For *Minio*, use `minio`.
- For *PostgreSQL*, use `user-provided-postgressql`.

You cannot use your own values for these secrets. The secrets must be created before you install the Speech services.

Deletions from `speech-override.yaml` file

The following entries have been removed from the `speech-override.yaml` file. They were added to work around a problem that has now been fixed.

```
sttRuntime:
  images:
    miniomc:
      tag:
        1.0.5
sttAMPatcher:
  images:
    miniomc:
      tag:
        1.0.5
ttsRuntime:
  images:
    miniomc:
      tag:
        1.0.5
```

The abbreviated `speech-override.yaml` file has generally been reduced further by fine-tuning its contents to the essential elements.

Version 1.2 (9 December 2020)

Version 1.2 is available

Speech to Text for IBM Cloud Pak for Data version 1.2 is now available. Installation and administration of the service include many changes. This version supports IBM Cloud Pak for Data versions 3.5 and 3.0.1, and Red Hat OpenShift versions 4.5 and 3.11.

New Australian and French Canadian models

The service now offers broadband and narrowband models for Australian English and Canadian French:

- Australian English: `en-AU_BroadbandModel` and `en-AU_NarrowbandModel`
- Canadian French: `fr-CA_BroadbandModel` and `fr-CA_NarrowbandModel`

The new models are generally available, and they support both language model and acoustic model customization.

- For more information about supported languages and models, see Previous-generation languages and models.
- For more information about language support for customization, see Language support for customization.

Updated models for improved speech recognition

The following language models have been updated for improved speech recognition:

- Brazilian Portuguese: `pt-BR_BroadbandModel` and `pt-BR_NarrowbandModel`
- French: `fr-FR_BroadbandModel`
- German: `de-DE_BroadbandModel` and `de-DE_NarrowbandModel`
- Japanese: `ja-JP_BroadbandModel`
- UK English: `en-GB_BroadbandModel` and `en-GB_NarrowbandModel`
- US English: `en-US_ShortForm_NarrowbandModel`

By default, the service automatically uses the updated models for all speech recognition requests. If you have custom language or custom acoustic models that are based on these models, you must upgrade your existing custom models to take advantage of the updates by using the following methods:

- `POST /v1/customizations/{customization_id}/upgrade_model`
- `POST /v1/acoustic_customizations/{customization_id}/upgrade_model`

For more information, see Upgrading custom models.

The `split_transcript_at_phrase_end` parameter is now generally available for all languages

The speech recognition parameter `split_transcript_at_phrase_end` is now generally available for all languages. Previously, it was generally available only for US and UK English. For more information, see Split transcript at phrase end .

Hesitation marker for German has changed

The hesitation marker that is used for the updated German broadband and narrowband models has changed from `[hesitation]` to `%HESITATION`. For more information about hesitation markers, see Speech hesitations and hesitation markers .

Defect fix: Address latency issue for models with large numbers of grammars

Defect fix: The service no longer has a latency issue for custom language models that contain a large number of grammars. When initially used for speech recognition, such custom models could take multiple seconds to load. The custom models now load much faster, greatly reducing latency when they are used for recognition.

15 July 2020 (Version 1.1.4)

Red Hat OpenShift version 4.3 is going out of service

IBM Cloud Pak for Data 3.0.1 is deprecating support for Red Hat OpenShift 4.3 on 1 September 2020. Red Hat OpenShift 4.3 is going out of service on **22 October 2020**. IBM Cloud Pak for Data is introducing support for Red Hat OpenShift 4.5. IBM Cloud Pak for Data is recommending that clients upgrade to Red Hat OpenShift 4.5 before 22 October 2020. IBM Support will work with any customers who already installed IBM Cloud Pak for Data 3.0.1 on Red Hat OpenShift 4.3. New customers who want to install on Red Hat OpenShift 4.x are instructed to install Red Hat OpenShift 4.5.

19 June 2020 (Version 1.1.4)

Version 1.1.4 is available

Speech to Text for IBM Cloud Pak for Data version 1.1.4 is now available. Installation and administration of the service include many changes. This version supports IBM Cloud Pak for Data versions 2.5 and 3.0.1, and Red Hat OpenShift versions 3.11 and 4.3. For more information about installing and managing the service, see Installing Watson Speech to Text version 1.1.4 .

New parameters to control the level of speech activity detection

The service now offers two new optional parameters for controlling the level of speech activity detection. The parameters can help ensure that only relevant audio is processed for speech recognition.

- The `speech_detector_sensitivity` parameter adjusts the sensitivity of speech activity detection. You can use the parameter to suppress word insertions from music, coughing, and other non-speech events.
- The `background_audio_suppression` parameter suppresses background audio based on its volume to prevent it from being transcribed or otherwise interfering with speech recognition. You can use the parameter to suppress side conversations or background noise.

You can use the parameters individually or together. They are available for all interfaces and for most language models. For more information about the parameters, their allowable values, and their effect on the quality and latency of speech recognition, see Speech activity detection.

New broadband and narrowband models for Dutch and Italian

The service now supports broadband and narrowband models for the Dutch and Italian languages:

- Dutch broadband model (`nl-NL_BroadbandModel`)
- Dutch narrowband model (`nl-NL_NarrowbandModel`)
- Italian broadband model (`it-IT_BroadbandModel`)
- Italian narrowband model (`it-IT_NarrowbandModel`)

Dutch and Italian language models are generally available (GA) for speech recognition and for language model and acoustic model customization. For more information about all available language models, see

- Supported previous-generation language models
- Language support for customization

Support for `speaker_labels` parameter for German and Korean

The service now supports speaker labels (the `speaker_labels` parameter) for German and Korean language models. Speaker labels identify which individuals spoke which words in a multi-participant exchange. For more information, see Speaker labels.

Improved speech recognition for Japanese narrowband model

The Japanese narrowband model (`ja-JP_NarrowbandModel`) now includes some multigram word units for digits and decimal fractions. The service returns these multigram units regardless of whether you enable smart formatting. The smart formatting feature understands and returns the multigram units that the model generates. If you apply your own post-processing to transcription results, you need to handle these units appropriately. For more information, see Japanese in the smart formatting documentation.

Simplified backup and restore

The service now offers greatly improved backup and restore procedures. Utilities are now available to back up data from your datastores, so you no longer need to re-create all of your data in the event of a disaster. For more information, see Backing up and restoring your data.

1 April 2020 (Version 1.1.3)

Acoustic model customization is now generally available

Acoustic model customization is now generally available (GA) for all supported languages. For more information about support for individual language models, see Language support for customization.

28 February 2020 (Version 1.1.3)

Version 1.1.3 is available

Speech to Text for IBM Cloud Pak for Data version 1.1.3 is now available.

New `end_of_phrase_silence_time` parameter

For speech recognition, the service now supports the `end_of_phrase_silence_time` parameter. The parameter specifies the duration of the pause interval at which the service splits a transcript into multiple final results. Each final result indicates a pause or extended silence that exceeds the pause interval. For most languages, the default pause interval is 0.8 seconds; for Chinese the default interval is 0.6 seconds.

You can use the parameter to effect a trade-off between how often a final result is produced and the accuracy of the transcription. Increase the interval when accuracy is more important than latency. Decrease the interval when the speaker is expected to say short phrases or single words.

For more information, see End of phrase silence time.

New `split_transcript_at_phrase_end` parameter

For speech recognition, the service now supports the `split_transcript_at_phrase_end` parameter. The parameter directs the service to split the transcript into multiple final results based on semantic features of the input, such as at the conclusion of sentences. The service bases its understanding of semantic features on the base language model that you use with a request. Custom language models and grammars can also influence how and where the service splits a transcript.

The parameter causes the service to add an `end_of_utterance` field to each final result to indicate the motivation for the split: `full_stop`, `silence`, `end_of_data`, or `reset`.

For more information, see Split transcript at phrase end.

Improved `speaker_labels` parameter

For speech recognition, the `speaker_labels` parameter has been updated to improve the identification of individual speakers for further analysis of your audio sample. For more information about the speaker labels feature, see Speaker labels. For more information about the improvements to the feature, see IBM Research AI Advances Speaker Diarization in Real Use Cases.

27 November 2019 (Version 1.1.2)

Version 1.1.2 is available

Speech to Text for IBM Cloud Pak for Data version 1.1.2 is now available.

Maximum number of custom models

You can create no more than 1024 custom language models and no more than 1024 custom acoustic models per owning credentials. For more information, see Maximum number of custom models.

30 August 2019 (Version 1.0.1)

Version 1.0.1 is available

Speech to Text for IBM Cloud Pak for Data version 1.0.1 is now available. The service now works with IBM Cloud Pak for Data 2.1.0.1. The service now supports installing IBM Cloud Pak for Data with Red Hat OpenShift.

New broadband and narrowband models for Spanish dialects

The service now offers broadband and narrowband language models in six Spanish dialects:

- Argentinian Spanish (`es-AR_BroadbandModel` and `es-AR_NarrowbandModel`)
- Castilian Spanish (`es-ES_BroadbandModel` and `es-ES_NarrowbandModel`)
- Chilean Spanish (`es-CL_BroadbandModel` and `es-CL_NarrowbandModel`)
- Colombian Spanish (`es-CO_BroadbandModel` and `es-CO_NarrowbandModel`)
- Mexican Spanish (`es-MX_BroadbandModel` and `es-MX_NarrowbandModel`)
- Peruvian Spanish (`es-PE_BroadbandModel` and `es-PE_NarrowbandModel`)

The Castilian Spanish models are not new. They are generally available for speech recognition and language model customization, and beta for acoustic model customization.

The models for the other five dialects are new and are beta for all uses. Because they are beta, these additional dialects might not be ready for production use and are subject to change. They are initial offerings that are expected to improve in quality with time and usage.

For more information, see the following sections:

- Supported previous-generation language models
- Language support for customization

FISMA support

Federal Information Security Management Act (FISMA) support is now available for Speech to Text for IBM Cloud Pak for Data. The service is FISMA

High Ready.

28 June 2019 (Version 1.0.0)

Version 1.0.0 is available

Version 1.0.0, the initial release of the service, is now available. Speech to Text for IBM Cloud Pak for Data is based on the IBM Watson® Speech to Text service on the public IBM Cloud. Speech to Text for IBM Cloud Pak for Data differs from the public Speech to Text service in the following ways. You might find this information helpful if you are already familiar with the Speech to Text service on the public IBM Cloud.

- Speech to Text for IBM Cloud Pak for Data uses access tokens for authentication. For more information, see the API & SDK reference.
- The endpoints for Speech to Text for IBM Cloud Pak for Data are specific to your IBM Cloud Pak for Data cluster. For more information, see the API & SDK reference.
- Speech to Text for IBM Cloud Pak for Data does not perform any request logging. You do not need to use the `X-Watson-Learning-Opt-Out` request header.
- Speech to Text for IBM Cloud Pak for Data does not support Watson tokens. You cannot use the `X-Watson-Authorization-Token` request header to authenticate with the service.

Using languages and models

Previous-generation languages and models

Starting **August 1, 2023**, all previous-generation models are now **discontinued** from the service. New clients must now only use the next-generation models. All existing clients must now migrate to the equivalent next-generation model. For more information, see Migrating to next-generation models.

The IBM Watson® Speech to Text service supports speech recognition with previous-generation models in many languages. The model indicates the language in which the audio is spoken and the rate at which it is sampled.

The models described on this page are referred to as *previous-generation models*. The service also offers *next-generation models* with enhanced qualities for improved speech recognition. For more information, see Next-generation languages and models.

Previous-generation model types

For most languages, the service makes available two types of previous-generation models:

- *Narrowband models* are intended for audio that has a minimum sampling rate of 8 kHz. Use narrowband models for offline decoding of telephone speech, which is the typical use for this sampling rate.
- *Broadband models* are for intended audio that has a minimum sampling rate of 16 kHz. Use broadband models for responsive, real-time applications, for example, for live-speech applications.

Choosing the correct model for your application is important. Use the model that matches the sampling rate (and language) of your audio. The service automatically adjusts the sampling rate of your audio to match the model that you specify. To achieve the best recognition accuracy, you also need to consider the frequency content of your audio. For more information, see Sampling rate and Audio frequency.

Supported previous-generation language models

The following sections list the previous-generation models of each type that are available for each language. The tables in the sections provide the following information:

- The *Model name* column indicates the name of the model.
- The *Status* column indicates whether the model is generally available (*GA*) or *Beta*.
- The *Recommended next-generation model* identifies the next-generation model that you can use instead of a deprecated model.

Note: Currently, not all broadband models have equivalent multimedia models. In such cases, consider using the telephony model for that language. The service downsamples the audio to the rate of the model that you use. So sending broadband audio to a telephony model might prove a sufficient alternative in cases where no equivalent multimedia model is currently available.

All models are available for both product versions, IBM Cloud and IBM Cloud Pak for Data.

Narrowband models

Table 1 lists the previous-generation narrowband models that are available.

Language	Model name	Status	Recommended next-generation model
Chinese (Mandarin)	`zh-CN_NarrowbandModel`	GA Discontinued	`zh-CN_Telephony`
Dutch (Netherlands)	`nl-NL_NarrowbandModel`	GA Discontinued	`nl-NL_Telephony`
English (Australian)	`en-AU_NarrowbandModel`	GA Discontinued	`en-AU_Telephony`
English (United Kingdom)	`en-GB_NarrowbandModel`	GA Discontinued	`en-GB_Telephony`
English (United States)	`en-US_NarrowbandModel`	GA Discontinued	`en-US_Telephony`

	`en-US_ShortForm_NarrowbandModel`	GA Discontinued	`en-US_Telephony`
French (Canadian)	`fr-CA_NarrowbandModel`	GA Discontinued	`fr-CA_Telephony`
French (France)	`fr-FR_NarrowbandModel`	GA Discontinued	`fr-FR_Telephony`
German	`de-DE_NarrowbandModel`	GA Discontinued	`de-DE_Telephony`
Italian	`it-IT_NarrowbandModel`	GA Discontinued	`it-IT_Telephony`
Japanese	`ja-JP_NarrowbandModel`	GA Discontinued	`ja-JP_Telephony` IBM Cloud
Korean	`ko-KR_NarrowbandModel`	GA Discontinued	`ko-KR_Telephony`
Portuguese (Brazilian)	`pt-BR_NarrowbandModel`	GA Discontinued	`pt-BR_Telephony`
Spanish (Argentinian, Beta)	`es-AR_NarrowbandModel`	Beta Discontinued	`es-LA_Telephony`
Spanish (Castilian)	`es-ES_NarrowbandModel`	GA Discontinued	`es-ES_Telephony`
Spanish (Chilean, Beta)	`es-CL_NarrowbandModel`	Beta Discontinued	`es-LA_Telephony`
Spanish (Colombian, Beta)	`es-CO_NarrowbandModel`	Beta Discontinued	`es-LA_Telephony`
Spanish (Mexican, Beta)	`es-MX_NarrowbandModel`	Beta Discontinued	`es-LA_Telephony`
Spanish (Peruvian, Beta)	`es-PE_NarrowbandModel`	Beta Discontinued	`es-LA_Telephony`

Table 1. Supported previous-generation narrowband models

Broadband models

Table 2 lists the previous-generation broadband models that are available.

Language	Model name	Status	Recommended next-generation model
Arabic (Modern Standard)	`ar-MS_BroadbandModel`	GA Discontinued	`ar-MS_Telephony`
Chinese (Mandarin)	`zh-CN_BroadbandModel`	GA Discontinued	`zh-CN_Telephony`
Dutch (Netherlands)	`nl-NL_BroadbandModel`	GA Discontinued	`nl-NL_Multimedia`
English (Australian)	`en-AU_BroadbandModel`	GA Discontinued	`en-AU_Multimedia`

English (United Kingdom)	`en-GB_BroadbandModel`	GA Discontinued	`en-GB_Multimedia`
English (United States)	`en-US_BroadbandModel`	GA Discontinued	`en-US_Multimedia`
French (Canadian)	`fr-CA_BroadbandModel`	GA Discontinued	`fr-CA_Multimedia`
French (France)	`fr-FR_BroadbandModel`	GA Discontinued	`fr-FR_Multimedia`
German	`de-DE_BroadbandModel`	GA Discontinued	`de-DE_Multimedia`
Italian	`it-IT_BroadbandModel`	GA Discontinued	`it-IT_Multimedia`
Japanese	`ja-JP_BroadbandModel`	GA Discontinued	`ja-JP_Multimedia`
Korean	`ko-KR_BroadbandModel`	GA Discontinued	`ko-KR_Multimedia`
Portuguese (Brazilian)	`pt-BR_BroadbandModel`	GA Discontinued	`pt-BR_Multimedia`
Spanish (Argentinian, Beta)	`es-AR_BroadbandModel`	Beta Discontinued	`es-LA_Telephony`
Spanish (Castilian)	`es-ES_BroadbandModel`	GA Discontinued	`es-ES_Multimedia`
Spanish (Chilean, Beta)	`es-CL_BroadbandModel`	Beta Discontinued	`es-LA_Telephony`
Spanish (Colombian, Beta)	`es-CO_BroadbandModel`	Beta Discontinued	`es-LA_Telephony`
Spanish (Mexican, Beta)	`es-MX_BroadbandModel`	Beta Discontinued	`es-LA_Telephony`
Spanish (Peruvian, Beta)	`es-PE_BroadbandModel`	Beta Discontinued	`es-LA_Telephony`

Table 2. Supported previous-generation broadband models

The US English short-form model (Deprecated)

The US English short-form model, `en-US_ShortForm_NarrowbandModel`, can improve speech recognition for Interactive Voice Response (IVR) and Automated Customer Support solutions. The short-form model is trained to recognize the short utterances that are frequently expressed in customer support settings like automated support call centers. In addition to being tuned for short utterances in general, the model is also tuned for precise utterances such as digits, single-character word and name spellings, and yes-no responses.

The `en-US_ShortForm_NarrowbandModel` is optimal for the kinds of responses that are common to human-to-machine exchanges, such as the use case of IBM® Voice Agent with Watson. The `en-US_NarrowbandModel` is generally optimal for human-to-human conversations. However, depending on the use case and the nature of the exchange, some users might find the short-form model suitable for human-to-human conversations as well. Given this flexibility and overlap, you might experiment with both models to determine which works best for your application. In either case, applying a custom language model with a grammar to the short-form model can further improve recognition results.

As with all models, noisy environments can adversely impact the results. For example, background acoustic noise from airports, moving vehicles, conference rooms, and multiple speakers can reduce transcription accuracy. Audio from speaker phones can also reduce accuracy due to the echo common to such devices. Using the parameters available for speech activity detection can counteract such effects and help improve speech transcription accuracy. Applying a custom acoustic model can further fine-tune the acoustics for speech recognition, but only as a final measure.

- For more information about language model and acoustic model customization, see Understanding customization.
- For more information about grammars, see Using grammars with custom language models.
- For more information about speech activity detection parameters, see Speech activity detection.

Supported features for previous-generation models

Previous-generation models are supported for use with almost all of the service's features. Most features and models are generally available for production use. Where indicated, some features and models are beta functionality. Restrictions apply for some features, for example:

- Features such as speaker labels, numeric redaction, and profanity filtering are limited to certain languages and models. Such restrictions are noted with the descriptions of the individual features. For more information about all available speech recognition parameters, see the Parameter summary.
- The `low_latency` parameter is supported only for next-generation models. For more information, see Low latency.
- For more information about previous-generation models' support for customization, see Customization support for previous-generation models.

Otherwise, when a feature is described as being available in general or available for a specific language or languages, it supports the previous-generation models.

Next-generation languages and models

The IBM Watson® Speech to Text service supports a growing collection of next-generation models that improve upon the speech recognition capabilities of the service's previous-generation models. The model indicates the language in which the audio is spoken and the rate at which it is sampled. Next-generation models have higher throughput than the previous-generation models, so the service can return transcriptions more quickly. Next-generation models also provide noticeably better transcription accuracy.

When you use next-generation models, the service analyzes audio bidirectionally. Using deep neural networks, the model analyzes and extracts information from the audio. The model then evaluates the information forwards and backwards to predict the transcription, effectively "listening" to the audio twice.

With the additional information and context afforded by bidirectional analysis, the service can make smarter hypotheses about the words spoken in the audio. Despite the added analysis, recognition with next-generation models is more efficient than with previous-generation models, so the service delivers results faster and with more accuracy. Most next-generation models also offer a low-latency option to receive results even faster, though low latency might impact transcription accuracy.

In addition to providing greater transcription accuracy, the models have the ability to hypothesize words that are not in the base language model and that it has not encountered in training. This capability can decrease the need for customization of domain-specific terms. A model does not need to contain a specific vocabulary term to predict that word.

- For an overview of the next-generation models and their technology, see Next-Generation Watson Speech to Text.
- For more information about the technology that underlies the next-generation models, see Advancing RNN Transducer Technology for Speech Recognition.
- For information about migrating from previous-generation to next-generation models, see Migrating to next-generation models.

Next-generation model types

The service makes available two types of next-generation models:

- Telephony models are intended specifically for audio that is communicated over a telephone. Like previous-generation *narrowband* models, telephony models are intended for audio that has a minimum sampling rate of 8 kHz.
- Multimedia models are intended for audio that is extracted from sources with a higher sampling rate, such as video. Use a multimedia model for any audio other than telephonic audio. Like previous-generation *broadband* models, multimedia models are intended for audio that has a minimum sampling rate of 16 kHz.

Choose the model type that most closely matches the source and sampling rate of your audio. The service automatically adjusts the sampling rate of your audio to match the model that you specify. To achieve the best recognition accuracy, also consider the frequency content of your audio. For more information, see Sampling rate and Audio frequency.

Supported next-generation language models

The following sections list the next-generation models of each type that are available for each language. The tables in the sections provide the following information:

- The *Model name* column indicates the name of the model. (Unlike previous-generation models, next-generation models do not include the word `Model` in their names.)
- The *Low-latency support* column indicates whether the model supports the `low_latency` parameter for speech recognition. For more information, see Low latency.
- The *Status* column indicates whether the model is generally available (GA) or beta.

The *Model name* and *Low-latency support* columns indicate the product versions for which the model and low-latency are supported. Unless otherwise labeled as IBM Cloud or IBM Cloud Pak for Data , a model and low latency are supported for both versions of the service.

Telephony models

Table 1 lists the available next-generation telephony models.

Language	Model name	Low-latency support	Status
Arabic (Modern Standard)	`ar-MS_Telephony`	Yes	GA
Chinese (Mandarin)	`zh-CN_Telephony`	Yes	GA
Czech	`cs-CZ_Telephony`	Yes	GA
Dutch (Belgian)	`nl-BE_Telephony`	Yes	GA
Dutch (Netherlands)	`nl-NL_Telephony`	Yes	GA
English (Australian)	`en-AU_Telephony`	Yes	GA
English (Indian)	`en-IN_Telephony`	Yes	GA
English (United Kingdom)	`en-GB_Telephony`	Yes	GA
English (United States)	`en-US_Telephony`	Yes	GA
English (all supported dialects)	`en-WW_Medical_Telephony`	Yes	Beta
French (Canadian)	`fr-CA_Telephony`	Yes	GA
French (France)	`fr-FR_Telephony`	Yes	GA
German	`de-DE_Telephony`	Yes	GA
Hindi (Indian)	`hi-IN_Telephony`	Yes	GA
Italian	`it-IT_Telephony`	Yes	GA
Japanese	`ja-JP_Telephony`	Yes	GA

Korean	`ko-KR_Telephony`	Yes	GA
Portuguese (Brazilian)	`pt-BR_Telephony`	Yes	GA
Spanish (Castilian)	`es-ES_Telephony`	Yes	GA
Spanish (Argentinian, Chilean, Colombian, Mexican, and Peruvian)	`es-LA_Telephony`	Yes	GA
Swedish	`sv-SE_Telephony`	Yes	GA

Table 1. Next-generation telephony models

Note: The Latin American Spanish model, `es-LA_Telephony`, applies to all Latin American dialects. It is the equivalent of the previous-generation models that are available for the Argentinian, Chilean, Colombian, Mexican, and Peruvian dialects. If you used a previous-generation model for any of these Latin American dialects, use the `es-LA_Telephony` model to migrate to the equivalent next-generation model.

Multimedia models

Table 2 lists the available next-generation multimedia models.

Language	Model name	Low-latency support	Status
Dutch (Netherlands)	`nl-NL_Multimedia`	Yes	GA
English (Australian)	`en-AU_Multimedia`	Yes	GA
English (United Kingdom)	`en-GB_Multimedia`	Yes	GA
English (United States)	`en-US_Multimedia`	Yes	GA
French (Canadian)	`fr-CA_Multimedia`	Yes	GA
French (France)	`fr-FR_Multimedia`	Yes	GA
German	`de-DE_Multimedia`	Yes	GA
Italian	`it-IT_Multimedia`	Yes	GA
Japanese	`ja-JP_Multimedia`	Yes	GA
Korean	`ko-KR_Multimedia`	Yes	GA
Portuguese (Brazilian)	`pt-BR_Multimedia`	Yes	GA
Spanish (Castilian)	`es-ES_Multimedia`	Yes	GA

Table 2. Next-generation multimedia models

The English medical telephony model

The beta next-generation `en-WW_Medical_Telephony` understands terms from the medical and pharmacological domains. Use the model in situations where you need to transcribe common medical terminology such as medicine names, product brands, medical procedures, illnesses, types of doctor, or COVID-19-related terminology.

Common use cases include conversations between a patient and a medical provider (for example, a doctor, nurse, or pharmacist):

- "My head is hurting. I need an Ibuprofen, please."
- "Can you suggest an orthopedist who specializes in osteoarthritis?"
- "Can you please help me find an internist in Chicago?"

The new model is available for all supported English dialects: Australian, Indian, UK, and US. The new model supports language model customization and grammars as beta functionality. It supports most of the same parameters as the `en-US_Telephony` model, including `smart_formatting` for US English audio. In addition to those features listed in Supported features for next-generation models, the model does *not* support the following parameters: `profanity_filter`, `redaction`, and `speaker_labels`.

Supported features for next-generation models

The next-generation models are supported for use with a large subset of the service's speech recognition features. In cases where a supported feature is restricted to certain languages, the same language restrictions usually apply to both previous- and next-generation models.

- For more information about the parameters that you can use with next-generation models, including their language support and whether the parameters are GA or beta, see the Parameter summary.
- For more information about next-generation models' support for customization, see Customization support for next-generation models.

Next-generation models support all speech recognition parameters and headers *except* for the following:

- `acoustic_customization_id` (Next-generation models do not support acoustic model customization.)
- `keywords` and `keywords_threshold`
- `processing_metrics` and `processing_metrics_interval`
- `word_alternatives_threshold`

Next-generation models also support the following parameters, which are not available with previous-generation models:

- `character_insertion_bias`, which is supported by all next-generation models. For more information, see Character insertion bias.
- `low_latency`, which is supported by most next-generation models. For more information, see Low latency.

Next-generation models also differ from previous-generation models with respect to the following additional features:

- Next-generation models do not produce hesitation markers. They instead include the actual hesitations in transcription results. For more information, see Speech hesitations and hesitation markers.
- Next-generation models support automatic capitalization only for German models. Previous-generation models support automatic customization only for US English models. For more information, see Capitalization.

Large speech languages and models

The IBM Watson® Speech to Text service supports a growing collection of Large Speech Models (LSMs) that improve the speech recognition capabilities of the service's previous-generation models. The model name is the locale, which consists of the language code and the region or country code that is separated by a dash. For example, `en-US` is for English that is spoken in the United states. LSMs are large models. They have a large number of trainable parameters and are trained on large amounts of audio. Because of their large size, the large amounts of training material they are built on, and the state-of-the-art architecture and training recipe that is used to build them, these models deliver more transcription accuracy compared to the previous models available.

You can use these models for both Telephony use cases and Broadband use cases.

Supported large speech model languages

The following table lists the large speech models that are available for each language.

Language	Model name	Status
English (Australian)	`en-AU`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024

English (Indian)	`en-IN`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024
English (United Kingdom)	`en-GB`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024
English (United States)	`en-US`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024
French (Canadian)	`fr-CA`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024
French (France)	`fr-FR`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024
Japanese	`ja-JP`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024
Portuguese (Brazilian)	`pt-BR`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024
Portuguese (Portugal)	`pt-PT`	IBM Cloud 23 August 2024 IBM Cloud Pak for Data 23 August 2024
Spanish (Castilian)	`es-ES`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024
Spanish (Argentinian)	`es-AR`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024
Spanish (Chilean)	`es-CL`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024
Spanish (Colombian)	`es-CO`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024
Spanish (Mexican)	`es-MX`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024
Spanish (Peruvian)	`es-PE`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024

Table 1. Large speech models

Supported features for large speech models

The large speech models are supported for use with a large subset of the service's speech recognition features. In cases where a supported feature is restricted to certain languages, the same language restrictions usually apply to large speech models, previous-generation, and next-generation models.

- For more information about the parameters that you can use with large speech models, including their language support and whether the parameters are GA or beta, see the Parameter summary.
- For more information about large speech models' support for customization, see Customization support for large speech models.

Large speech models support all speech recognition parameters and headers *except*:

- `acoustic_customization_id` (Large speech models do not support acoustic model customization.)
- `keywords` and `keywords_threshold`
- `word_alternatives_threshold`
- `grammar_name` (Large speech models do not support grammar customization.)
- `low_latency` (Large speech models natively support low latency out of the box.)
- `character_insertion_bias`

Large speech models also differ from previous-generation models with respect to the following additional feature:

- Large speech models do not produce hesitation markers. They instead include the actual hesitations in transcription results. For more information, see Speech hesitations and hesitation markers.

Migrating to large speech models

Starting **August 1, 2023**, all previous-generation models are now **discontinued** from the service. New clients must now only use the large speech models or next-generation models. All existing clients must now migrate to the equivalent large speech model or next-generation model. For more information, see Migrating to large speech models.

You must migrate your use of any deprecated previous-generation models to the equivalent large speech models or next-generation models by the 31 July 2023 end of service date. Next-generation models provide appreciably better transcription accuracy and throughput. But they currently provide slightly fewer features than previous-generation models.

This topic provides an overview of the steps that you need to take to migrate from previous- to next-generation models. For more information about migrating, you can also see Watson Speech to Text: How to Plan Your Migration to the Next-Generation Models.

Step 1: Identify the large speech model or next-generation model to which to migrate

The following topics describe all large speech models, previous- and next-generation models:

- Previous-generation languages and models
- Next-generation languages and models
- Large speech languages and models

The tables in Supported previous-generation language models list the recommended large speech model or next-generation model to which to migrate from a previous generation model. Use the indicated large speech model or next-generation model in your speech recognition requests.

The service continues to make new large speech models available. All new models are identified in the release notes and in the tables that describe the available models.

Optimally, you migrate from a narrowband model / telephony model to a large speech model, and from a broadband model / multimedia model to a large speech model. However, not all broadband models and multimedia models have equivalent large speech models. In such cases, you can migrate from a narrowband model to a telephony model, or from a broadband model to a multimedia model. The service downsamples the audio that you send to the rate of the model that you use. So sending broadband audio to a telephony model might prove a sufficient alternative in cases where no equivalent multimedia model is currently available.

For example, the following speech recognition request uses the previous-generation `en-US_NarrowbandModel`:

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?model=en-US_NarrowbandModel"
```

To use the equivalent large speech model `en-US` , you simply change the value that you pass with the `model` query parameter:

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?model=en-US"
```

Step 2: Identify the features that are available with large speech models

Large speech models support slightly fewer features and parameters than previous- and next-generation models. However, although they lack full parity, most features are available with both types of models. And where a feature is limited to a subset of languages, the limitations apply equally to all types of models.

For information about the features that are supported with the different model types, see

- Supported features for previous-generation models
- Supported features for next-generation models
- Supported features for large speech models
- Parameter summary

The service continues to make new features available with next-generation models. All updates to feature support are documented in the release notes and in the documentation for the model types.

To migrate to next-generation models, you must remove features that aren't supported by next-generation models from your speech recognition requests. You can also consider using features such as character insertion bias that are available only with large speech models and next-generation models.

For example, the following speech recognition request uses the `profanity_filter`, `redaction`, and `word_alternatives_threshold` parameters with the previous-generation `en-US_NarrowbandModel`:

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path_to_file}audio-file.flac \
"{url}/v1/recognize?model=en-US_NarrowbandModel&profanity_filter=true&redaction=true&word_alternatives_threshold=0.50"
```

Only the `word_alternatives_threshold` parameter is not supported by large speech models. To use the equivalent large speech model `en-US_Telephony` model, you simply change the value that you pass with the `model` query parameter and eliminate the `word_alternatives_threshold` parameter:

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path_to_file}audio-file.flac \
"{url}/v1/recognize?model=en-US&profanity_filter=true&redaction=true"
```

Step 3: Re-create any custom language models that you use

You must re-create any custom language models that are based on previous-generation or next-generation models by basing them on the equivalent large speech models. This requires that you create a new custom language model and add your corpora, and custom words from the old model to the new model.

In general, large speech models do not rely as heavily on custom language models. They use a different approach to transcription that minimizes the need for language model customization.

Note: Large speech models do not support custom acoustic models. Because of how the models transcribe audio, acoustic model customization is not necessary.

For example, the following speech recognition request uses a custom language model that is based on the `en-US_NarrowbandModel`. In this example, the custom model has the identifier `8acf31fa-0aa2-4ecc-a805-1f527f342dba`.

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognize?model=en-US_NarrowbandModel&language_customization_id=8acf31fa-0aa2-4ecc-a805-1f527f342dba"
```

After you re-create the custom language model with the equivalent `en-US` model, simply update the model name to `en-US` and the `language_customization_ID` parameter to use the identifier of the new custom model, `636d8494-7e53-436a-8557-30d6b2a63cd7`:

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognize?model=en-US&language_customization_id=636d8494-7e53-436a-8557-30d6b2a63cd7"
```

Step 4: Evaluate the results of the large speech model

Once you have updated your speech recognition requests to use large speech models, eliminated unsupported parameters, and re-created any custom language models, you can experiment with speech recognition based on previous- and next-generation models. Compare the resulting transcripts to determine whether the large speech model produces equivalent or better results. Also consider the performance of requests that use large speech model to determine how much faster you receive results.

You can also compare the word error rate of the large speech models, previous- and next-generation results. The open-source Word Error Rate (WER) utility, which is available in Python, can help you measure and compare the accuracy of your results.

Using a model for speech recognition

You use the `model` parameter of a speech recognition request to indicate the model that is to be used with the request. You can specify a large speech model, previous- or next-generation model with the parameter.

For more information about the models that are available for speech recognition, see

- Previous-generation languages and models
- Next-generation languages and models
- Large speech languages and models

Specify a previous-generation model example

The following example HTTP request uses the previous-generation model `en-US_NarrowbandModel` for speech recognition:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?model=en-US_NarrowbandModel"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?model=en-US_NarrowbandModel"
```

Specify a next-generation model example

The following example HTTP request uses the next-generation `en-US_Telephony` model for speech recognition:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?model=en-US_Telephony"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?model=en-US_Telephony"
```

Specify a large speech model example

The following example HTTP request uses the large speech model `en-US` for speech recognition:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?model=en-US"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?model=en-US"
```

Using the default model

If you omit the `model` parameter from a speech recognition request, the service uses the US English `en-US_BroadbandModel` by default. This default applies to all speech recognition requests.

IBM Cloud Pak for Data If you do not install the `en-US_BroadbandModel`, it cannot serve as the default model. In this case, you must either

- Use the `model` parameter to pass the model that is to be used with each request.
- Specify a new default model for your installation of Speech to Text for IBM Cloud Pak for Data by using the `defaultSTTModel` property in the Speech services custom resource. For more information, see Installing Watson Speech to Text.

Listing information about models

The IBM Watson® Speech to Text service provides methods for listing information about all of its available models or about a specific large speech model, previous- or next-generation model.

Model information

Regardless of whether you list information about all available models or about a specific model, the service returns the same output for a model:

- `name` is the name of the model that you use in a request.
- `language` is the language identifier of the model (for example, `en-US`).
- `rate` identifies the minimum acceptable sampling rate in Hertz for audio that is used with the model.
- `url` is the URI for the model.
- `description` provides a brief description of the model.
- `supported_features` describes the additional service features that are supported with the model:
  - `custom_language_model` is a boolean that indicates whether you can create custom language models that are based on the model.
  - IBM Cloud `custom_acoustic_model` is a boolean that indicates whether you can create custom acoustic models that are based on the model.
  - `low_latency` is a boolean that indicates whether you can use the `low_latency` parameter with a next-generation model. The service includes this field only for next-generation models. Large speech models and previous-generation models do not support the `low_latency` parameter.
  - `speaker_labels` indicates whether you can use the `speaker_labels` parameter with the model.

Note: The `speaker_labels` field returns `true` for all models. However, speaker labels are supported as beta functionality for only a limited number of languages. For more information, see Speaker labels.

Listing all models

You use the HTTP `GET /v1/models` method to list information about all available models. The service returns an array of JSON objects that provides information about all of the models.

The order in which the service returns models can change from call to call. So, do not rely on an alphabetized or static list of models. Because the models are returned as an array of JSON objects, the order has no bearing on programmatic uses of the response.

List all models example

The following example lists all models that are supported by the service:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/models"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/models"
```

The response is abbreviated to show only the first few models.

```
{
  "models": [
    {
      "name": "pt-BR_NarrowbandModel",
      "language": "pt-BR",
      "url": "{url}/v1/models/pt-BR_NarrowbandModel",
      "rate": 8000,
      "supported_features": {
        "custom_language_model": true,
        "custom_acoustic_model": true,
        "speaker_labels": true
      },
      "description": "Brazilian Portuguese narrowband model."
    },
    {
      "name": "ko-KR_BroadbandModel",
      "language": "ko-KR",
      "url": "{url}/v1/models/ko-KR_BroadbandModel",
      "rate": 16000,
      "supported_features": {
        "custom_language_model": true,
        "custom_acoustic_model": true,
        "speaker_labels": true
      },
      "description": "Korean broadband model."
    },
    {
      "name": "fr-FR_BroadbandModel",
      "language": "fr-FR",
      "url": "{url}/v1/models/fr-FR_BroadbandModel",
      "rate": 16000,
      "supported_features": {
        "custom_language_model": true,
        "custom_acoustic_model": true,
        "speaker_labels": true
      },
      "description": "French broadband model."
    },
    . . .
  ]
}
```

Listing a specific model

You use the HTTP `GET /v1/models/{model_id}` method to list information about a specified model. The service returns information only for that model.

List a specific previous-generation model example

The following example shows information about the previous-generation US English broadband model, `en-US_BroadbandModel`:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/models/en-US_BroadbandModel"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/models/en-US_BroadbandModel"
```

The model supports language model customization, acoustic model customization, and speakers labels.

```
{
  "rate": 16000,
  "name": "en-US_BroadbandModel",
  "language": "en-US",
  "url": "{url}/v1/models/en-US_BroadbandModel",
  "supported_features": {
    "custom_language_model": true,
    "custom_acoustic_model": true,
    "speaker_labels": true
  },
  "description": "US English broadband model."
}
```

List a specific next-generation model example

The following example shows information about the next-generation US English telephony model, `en-US_Telephony`:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/models/en-US_Telephony"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/models/en-US_Telephony"
```

The model supports low latency, speakers labels, and language model customization. It does not support acoustic model customization.

```
{
   "name": "en-US_Telephony",
   "rate": 8000,
   "language": "en-US",
   "description": "US English telephony model for narrowband audio (8kHz)",
   "supported_features": {
      "custom_language_model": true,
      "custom_acoustic_model": false,
      "low_latency": true,
      "speaker_labels": true
   },
   "url": "{url}/v1/models/en-US_Telephony"
}
```

List a specific large speech model example

The following example shows information about the English large speech model, `en-US`:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/models/en-US"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/models/en-US"
```

The model supports speakers labels, and language model customization. It does not support acoustic model customization and low latency.

```
$ {
  "name": "en-US",
  "rate": 8000,
  "language": "en-US",
  "description": "Large US English model",
  "supported_features": {
    "custom_language_model": true,
    "custom_acoustic_model": false,
    "low_latency": false,
    "speaker_labels": true
    },
  "url": "{url}/v1/models/en-US"
}
```

Using audio formats

Audio terminology and characteristics

The following terminology is used to describe the characteristics of audio data and its processing. This information is helpful for using your audio with the IBM Watson® Speech to Text service.

- If you are unfamiliar with audio and how it is described and specified, begin with this topic to help you get started.
- If you already understand how to work with audio data, start with Supported audio formats.

Sampling rate

Sampling rate (or sampling frequency) is the number of audio samples that are taken per second. Sampling rate is measured in Hertz (Hz) or kilohertz (kHz). For example, a rate of 16,000 samples per second is equal to 16,000 Hz (or 16 kHz). With the Speech to Text service, you specify a model to indicate the sampling rate of your audio:

- *Broadband* and *multimedia* models are used for audio that is sampled at no less than 16 kHz, which IBM® recommends for responsive, real-time applications (for example, for live-speech applications).
- *Narrowband* and *telephony* models are used for audio that is sampled at no less than 8 kHz, which is the rate that is typically used for telephonic audio.

The service supports both sampling rates for most languages and formats. It automatically adjusts the sampling rate of your audio to match the model that you specify before it recognizes speech.

- For broadband and multimedia models, the service converts audio recorded at higher sampling rates to 16 kHz.
- For narrowband and telephony models, it converts audio recorded at higher sampling rates to 8 kHz.

You can, for instance, send 44 kHz audio with any model, but that needlessly increases the size of the audio. To maximize the amount of audio that you can send, match the sampling rate of your audio to the model that you use.

The service does not accept audio that is sampled at a rate that is less than the sampling rate of the model. For example, you cannot use a broadband or multimedia model to recognize audio that is sampled at a rate of 8 kHz.

Notes about audio formats

- For the `audio/alaw`, `audio/l16`, and `audio/mulaw` formats, you must specify the rate of your audio.
- For the `audio/basic` and `audio/g729` formats, the service supports only narrowband audio.

More information

- For more information about sampling rates, see Sampling (signal processing).
- For more information about the models that the service offers for each supported language, see Large speech languages and models, Previous-generation languages and models and Next-generation languages and models.

Bit rate

Bit rate is the number of bits of data that is sent per second. The bit rate for an audio stream is measured in kilobits per second (kbps). The bit rate is calculated from the sampling rate and the number of bits stored per sample. For speech recognition, IBM® recommends that you record 16 bits per sample for your audio.

For example, audio that uses a broadband sampling rate of 16 kHz and 16 bits per sample has a bit rate of 256 kbps: `(16,000 * 16) / 1000`.

More information

- For more information about bit rates, see Bit rate.
- For a general discussion of sampling rates and bit rates, see What are bit rates? and Choosing bit rates for podcasts.

Compression

Compression is used by many audio formats to reduce the size of the audio data. Compression reduces the number of bits stored per sample and thus the bit rate. Some formats use no compression, but most offer one of the two basic types:

- *Lossless* compression reduces the size of the audio with no loss of quality, but the compression ratio is typically small.
- *Lossy* compression reduces the size of the audio by as much as 10 times, but some data and some quality is irretrievably lost in the compression.

You can use compression to accommodate more audio data with your speech recognition request. But the type of compression that you use has implications for transcription quality.

Notes about audio formats

- The `audio/ogg` and `audio/webm` formats are containers whose compression relies on the codec that you use to encode the data: Opus or Vorbis.
- The `audio/wav` format is a container that can include uncompressed, lossless, or lossy data.

More information

- For more information about audio compression, see Data compression (Audio).
- For more information about the compression that is available with the audio formats that the service supports, see Audio formats.
- For more information about using data compression to increase the amount of audio that you can send with a request, see Data limits and compression.

Channels

Channels indicate the number of streams of the recorded audio:

- *Monaural* (or mono) audio has only a single channel.
- *Stereophonic* (or stereo) audio typically has two channels.

The Speech to Text service accepts audio with a maximum of 16 channels. Because it uses only a single channel for speech recognition, the service downmixes audio with multiple channels to one-channel mono during transcoding.

Notes about audio formats

- For the `audio/l16` format, you must specify the number of channels if your audio has more than one channel.
- For the `audio/wav` format, the service accepts audio with a maximum of nine channels.

More information

- For more information about audio channels, see Audio signal.

Endianness

Endianness indicates how bytes of data are organized by the underlying computer architecture:

- *Big-endian* orders data by most-significant bit.
- *Little-endian* orders data by least-significant bit.

The Speech to Text service automatically detects the endianness of the incoming audio.

Notes about audio formats

- For the `audio/l16` format, you can specify the endianness to disable auto-detection if needed.

More information

- For more information about endianness, see Endianness.

Audio frequency

Audio frequency refers to the range of audible frequencies in the audio. The standard audible frequency for humans is generally accepted as 20 to 20,000 Hz. You can use spectrographic analysis to produce a spectrogram that reveals the frequency content of your audio.

The sampling rate that is applied to audio is typically twice the maximum frequency of the audio. For example, a sampling rate of 16 kHz means that the maximum frequency of the sampled audio signal is 8 kHz. The service's models are created with this is mind.

- The narrowband models are built with audio that is sampled at 8 kHz. Narrowband models expect to find information in a range that is less than or equal to 4 kHz.
- The broadband models are built with audio that is sampled at 16 kHz. Broadband models expect to find information in the 4-8 kHz range.

The training data for the models is derived from different channels (telephony for narrowband models). The models reflect the characteristics of the channels on which they were trained.

Upsampling

Upsampling increases the sampling rate of the audio but introduces no new information into the audio. It produces an approximation of the audio signal that would have been obtained by sampling the audio at a higher rate. It increases the size of the audio data.

The information in audio that is originally sampled at a narrowband frequency is limited to the 0-4 kHz range. Upsampling narrowband audio to a higher sampling rate is unlikely to improve speech recognition accuracy. If you upsample narrowband audio, it lacks information in the range that the broadband models expect. Furthermore, the information that is found in the expected range of a narrowband sample is qualitatively different from the information that is found in the same range of a broadband sample. So upsampling actually results in degraded recognition accuracy.

For a broadband sampling rate of 16 kHz, the maximum frequency present in the sampled audio signal is expected to be 8 kHz. Therefore, you must filter the original signal at 8 kHz before you sample it with a rate of 16 kHz. Otherwise, degradation occurs due to the phenomenon known as *aliasing*. To understand why, see Nyquist frequency.

A useful comparison might be to imagine viewing a VHS tape on a large flat-screen HDTV. The image is blurry because playing the tape on a high-definition device cannot really add new information to the stream. It simply makes the format compatible with the better device. The same is true of upsampling audio.

Downsampling

Downsampling decreases the sampling rate of the audio. It produces an approximation of the audio signal that would have been obtained by sampling the audio at a lower rate. Downsampling removes no information from the audio signal, but it does reduce the size of the audio data.

Downsampling your audio can be effective in some cases. For example, if the sampling rate of your audio is greater than 8 kHz *and* a spectrographic examination reveals no frequency content that is greater than 4 kHz, consider downsampling the audio to 8 kHz.

More information

- For more information about audio frequency, see Audio frequency.
- For more information about upsampling, see Upsampling.
- For more information about downsampling, see Downsampling.

Supported audio formats

The IBM Watson® Speech to Text service can extract speech from audio in many formats. Later sections of this topic can help you get the most from your use of the service. If you are unfamiliar with audio processing, start with Audio terminology and characteristics for information about audio concepts.

Audio formats

Table 1 provides a summary of the audio formats that the service supports.

- *Audio format* identifies each supported format by its `Content-Type` specification.
- *Compression* indicates the format's support for compression. By using a format that supports compression, you can reduce the size of your audio to maximize the amount of data that you can pass to the service. But you need to consider the possible effects of compression on the quality of the audio. For more information, see Data limits and compression .
- *Content-type specification* indicates whether you must use the `Content-Type` header or equivalent parameter to specify the format (MIME type) of the audio that you send to the service. For more information, see Specifying an audio format.

The final columns identify additional *Required parameters* and *Optional parameters* for each format. The following sections provide more information about these parameters.

Audio format	Compression	Content-type specification	Required parameters	Optional parameters
audio/alaw	Lossy	Required	`rate={integer}`	None
audio/basic	Lossy	Required	None	None
audio/flac	Lossless	Optional	None	None
audio/g729	Lossy	Optional	None	None

audio/l16	None	Required	`rate={integer}`	`channels={integer}` `endianness=big-endian` `endianness=little-endian`
audio/mp3 audio/mpeg	Lossy	Optional	None	None
audio/mulaw	Lossy	Required	`rate={integer}`	None
audio/ogg	Lossy	Optional	None	`codecs=opus` `codecs=vorbis`
audio/wav	None, lossless, or lossy	Optional	None	None
audio/webm	Lossy	Optional	None	`codecs=opus` `codecs=vorbis`

Table 1. Summary of supported audio formats

audio/alaw format

A-law (`audio/alaw`) is a single-channel, lossy audio format. It uses an algorithm that is similar to the u-law algorithm applied by the `audio/basic` and `audio/mulaw` formats, though the A-law algorithm produces different signal characteristics. When you use this format, the service requires an extra parameter on the format specification.

Parameter	Description
`rate` *Required*	An integer that specifies the sampling rate at which the audio is captured. For example, specify the following parameter for audio data that is captured at 8 kHz: `audio/alaw;rate=8000`

Table 2. Parameter for audio/alaw format

For more information, see A-law algorithm.

audio/basic format

Basic audio (`audio/basic`) is a single-channel, lossy audio format that is encoded by using 8-bit u-law (or mu-law) data that is sampled at 8 kHz. This format provides a lowest-common denominator for indicating the media type of audio. The service supports the use of files in `audio/basic` format only with narrowband models.

For more information, see the Internet Engineering Task Force (IETF) Request for Comment (RFC) 2046 and iana.org/assignments/media-types/audio/basic.

audio/flac format

Free Lossless Audio Codec (FLAC) (`audio/flac`) is a lossless audio format. For more information, see FLAC.

audio/g729 format

G.729 (`audio/g729`) is a lossy audio format that supports data that is encoded at 8 kHz. The service supports only G.729 Annex D, not Annex J. The service supports the use of files in `audio/g729` format only with narrowband models. For more information, see G.729.

audio/l16 format

Linear 16-bit Pulse-Code Modulation (PCM) (`audio/l16`) is an uncompressed audio format. Use this format to pass a raw PCM file. Linear PCM audio can also be carried inside of a container Waveform Audio File Format (WAV) file. When you use the `audio/l16` format, the service accepts extra required and optional parameters on the format specification.

Parameter	Description

`rate` *Required*	An integer that specifies the sampling rate at which the audio is captured. For example, specify the following parameter for audio data that is captured at 16 kHz: `audio/l16;rate=16000`
`channels` *Optional*	By default, the service treats the audio as if it has a single channel. *If the audio has more than one channel,* you must specify an integer that identifies the number of channels. For example, specify the following parameter for two-channel audio data that is captured at 16 kHz: `audio/l16;rate=16000;channels=2` The service accepts a maximum of 16 channels. It downmixes the audio to one channel during transcoding.
`endianness` *Optional*	By default, the service auto-detects the endianness of incoming audio. But its auto-detection can sometimes fail and drop the connection for short audio in **`audio/l16`** format. Specifying the endianness disables auto-detection. Specify either **`big-endian`** or **`little-endian`**. For example, specify the following parameter for audio data that is captured at 16 kHz in little-endian format: `audio/l16;rate=16000;endianness=little-endian` Section 5.1 of [Request for Comment (RFC) 2045](#) specifies big-endian format for `audio/l16` data, but many people use little-endian format.

Table 3. Parameters for audio/l16 format

For more information, see the IETF [Request for Comment (RFC) 2586](#) and [Pulse-code modulation](#).

audio/mp3 and audio/mpeg formats

MP3 (`audio/mp3`) or *Motion Picture Experts Group (MPEG)* (`audio/mpeg`) is a lossy audio format. (MP3 and MPEG refer to the same format.) For more information, see [MP3](#).

audio/mulaw format

Mu-law (`audio/mulaw`) is a single-channel, lossy audio format. The data is encoded by using the u-law (or mu-law) algorithm. The `audio/basic` format is an equivalent format that is always sampled at 8 kHz. When you use this format, the service requires an extra parameter on the format specification.

Parameter	Description
`rate` *Required*	An integer that specifies the sampling rate at which the audio is captured. For example, specify the following parameter for audio data that is captured at 8 kHz: `audio/mulaw;rate=8000`

Table 4. Parameter for audio/mulaw format

For more information, see [M-law algorithm](#).

audio/ogg format

Ogg (`audio/ogg`) is an open container format that is maintained by the Xiph.org Foundation ([xiph.org/ogg](#)). You can use audio streams that are compressed with the following lossy codecs:

- *Opus* (`audio/ogg;codecs=opus`). For more information, see [opus-codec.org](#) and [Opus (audio format)](#). Look especially at the *Containers* section.
- *Vorbis* (`audio/ogg;codecs=vorbis`). For more information, see [xiph.org/vorbis](#) and [Vorbis](#).

OGG Opus is the preferred codec. It is the logical successor to OGG Vorbis because of its low latency, high audio quality, and reduced size. It is standardized by the Internet Engineering Task Force (IETF) as [Request for Comment (RFC) 6716](#).

If you omit the codec from the content type, the service automatically detects it from the input audio.

audio/wav format

Waveform Audio File Format (WAV) (`audio/wav`) is a container format that is often used for uncompressed audio streams, but it can contain compressed audio, as well. The service supports WAV audio that uses any encoding. It accepts WAV audio with a maximum of nine channels (due to an FFmpeg limitation).

For more information about the WAV format, see [WAV](#). For more information about reducing the size of WAV audio by converting it to the Opus codec, see [Converting to audio/ogg with the Opus codec](#).

audio/webm format

Web Media (WebM) (`audio/webm`) is an open container format that is maintained by the WebM project (webmproject.org). You can use audio streams that are compressed with the following lossy codecs:

- *Opus* (`audio/webm;codecs=opus`). For more information, see opus-codec.org and Opus (audio format). Look especially at the *Containers* section.
- *Vorbis* (`audio/webm;codecs=vorbis`). For more information, see xiph.org/vorbis and Vorbis.

If you omit the codec, the service automatically detects it from the input audio.

Tip: For JavaScript code that shows how to capture audio from a microphone in a Chrome browser and encode it into a WebM data stream, see jsbin.com/hedujihuqo/edit?js,console. The code does not submit the captured audio to the service.

Specifying an audio format

You use the `Content-Type` request header or equivalent parameter to specify the format (MIME type) of the audio that you send to the service. You can specify the audio format for any request, but that's not always necessary:

- For most formats, the content type is optional. You can omit the content type or specify `application/octet-stream` to have the service automatically detect the format.
- For other formats, the content type is required. These formats do not provide the information, such as the sampling rate, that the service needs to auto-detect their format. You must specify a content type for the `audio/alaw` , `audio/basic` , `audio/l16` , and `audio/mulaw` formats.

For more information about the formats that require a content-type specification, see Table 1 in Audio formats. The table's *Content-type specification* column indicates whether you must specify the content type.

For examples of specifying a content type with each of the service's interfaces, see Making a speech recognition request . All of the examples in that topic specify a content type.

Important: When you use the `curl` command to make a speech recognition request with the HTTP interfaces, you must either specify the audio format with the `Content-Type` header, specify `"Content-Type: application/octet-stream"` , or specify just `"Content-Type:"` . If you omit the header entirely, `curl` uses a default value of `application/x-www-form-urlencoded` .

Data limits and compression

The service accepts a maximum of 100 MB of audio data for transcription with a synchronous HTTP or WebSocket request, and 1 GB of audio data with an asynchronous HTTP request. When you recognize long continuous audio streams or large audio files, you need to consider and accommodate these data limits.

One way to maximize the amount of audio data that you can pass with a speech recognition request is to use a format that offers Compression. There are two basic types of compression, lossy and lossless. The audio format and compression algorithm that you choose can have a direct impact on the accuracy of speech recognition.

Audio formats that use lossy compression significantly reduce the size of your audio stream. But compressing the audio too severely can result in lower transcription accuracy. You can't hear the difference, but the service is much more sensitive to such data loss.

Comparing approximate audio sizes

Consider the approximate size of the data stream that results from 2 hours of continuous speech transmission that is sampled at 16 kHz and at 16 bits per sample:

- If the data is encoded with the `audio/wav` format, the two-hour stream has a size of 230 MB, well beyond the service's 100 MB limit.
- If the data is encoded in the `audio/ogg` format, the two-hour stream has a size of only 23 MB, well beneath the service's limit.

The following table approximates the maximum duration of audio that can be sent for speech recognition with a synchronous HTTP or WebSocket request in different formats. The duration considers the 100 MB service limit. Actual values can vary depending on the complexity of the audio and the achieved compression rate.

Audio format	Maximum duration of audio (approximate)
`audio/wav`	55 minutes
`audio/flac`	1 hour 40 minutes

`audio/mp3`	3 hours 20 minutes
`audio/ogg`	8 hours 40 minutes

Table 5. Maximum duration of audio in different formats

In testing to compare the different audio formats, IBM® determined that the WAV and FLAC formats delivered the best word error rate (WER). These formats can serve as a baseline for transcription accuracy because they maintain the audio intact with no data loss. The Ogg format with the Opus coded showed a slight degradation of 2% WER relative to the baseline. The MP3 format delivered the worst results, with a 10% degradation of WER relative to the baseline.

Note: The `audio/ogg;codecs=opus` and `audio/webm;codecs=opus` formats are generally equivalent, and their sizes are almost identical. They use the same codec internally; only the container format is different.

Maximizing transcription accuracy

When choosing an audio format and compression algorithm, consider the following recommendations to maximize transcription accuracy:

- *Use an uncompressed and lossless audio format.* If the duration of your audio is less than 55 minutes (less than 100 MB), consider using the `audio/wav` format. Although the WAV format can accommodate only 55 minutes of audio, that is often sufficient for most transcription applications, such as customer support calls. And uncompressed WAV audio can produce more accurate transcription.
- *Use the asynchronous HTTP interface.* If you elect to use the WAV format but your audio exceeds the 100 MB limit, the asynchronous interface allows you to send up to 1 GB of data.
- *Use a compressed but lossless audio format.* If you have to compress your audio file, use the `audio/flac` format, which employs lossless compression. Lossless compression reduces the size of the audio but maintains its quality. The FLAC format is a good candidate for maximizing transcription accuracy.
- *Use lossy compression as a last resort.* If you need even greater compression, use the `audio/ogg` format with the Opus codec. Although the Ogg format uses lossy compression, the combination of the Ogg format with the Opus codec showed the least degradation in speech accuracy among lossy compression algorithms.

Using other formats with greater levels of compression can compromise the accuracy of transcription. Experiment with the service to determine which format is best for your audio and application. For more ways to improve speech recognition, see Tips for improving speech recognition .

Audio conversion

You can use various tools to convert your audio to a different format. The tools can be helpful when your audio is in a format that is not supported by the service or is in an uncompressed or lossless format. In the latter case, you can convert the audio to a lossy format to reduce its size.

The following freeware tools are available to convert your audio from one format to another:

- Sound eXchange (SoX) (sox.sourceforge.net).
- FFmpeg (ffmpeg.org). You can also use FFmpeg to separate audio from a multimedia file that contains both audio and video data. For more information, see Transcribing speech from video files .
- Audacity® (audacityteam.org).
- For Ogg format with the Opus codec, opus-tools.

These tools offer cross-platform support for multiple audio formats. Moreover, you can use many of the tools to play your audio. Do not use the tools to violate applicable copyright laws.

Converting to audio/ogg with the Opus codec

The opus-tools include three command-line utilities for working with Ogg audio in the Opus codec:

- The opusenc utility encodes audio from WAV, FLAC, and other formats to Ogg with the Opus codec. The page shows how to compress audio streams. Compression is useful for passing real-time audio to the service.
- The opusdec utility decodes audio from the Opus codec to uncompressed PCM WAV files.
- The opusinfo utility provides information about and validity checking for Opus files.

Many users send WAV files for speech recognition. With the service's 100 MB data limit for synchronous HTTP and WebSocket requests, the WAV format reduces the amount of audio that can be recognized with a single request. Using the **opusenc** command to convert the audio to the preferred `audio/ogg:codecs=opus` format can greatly increase the amount of audio that you can send with a recognition request.

For example, consider an uncompressed broadband (16 kHz) WAV file (**input.wav**) that uses 16 bits per sample for a bit rate of 256 kbps. The following command converts the audio to a file (**output.opus**) that uses the Opus codec:

```
$ opusenc input.wav output.opus
```

The conversion compresses the audio by a factor of four and produces an output file with a bit rate of 64 kbps. However, according to the Opus Recommended Settings, you can safely reduce the bit rate to 24 kbps and still retain a full band for speech audio. This reduction compresses the audio by a factor of 10. The following command uses the `--bitrate` option to produce an output file with a bit rate of 24 kbps:

```
$ opusenc --bitrate 24 input.wav output.opus
```

Compression with the **opusenc** utility is fast. Compression happens at a rate that is approximately 100 times faster than real time. When it finishes, the command writes output to the console that provides complete details about its running time and the resulting audio data.

Transcribing speech from video files

You cannot transcribe speech from a multimedia file that contains both audio and video. The service accepts only audio data for speech recognition.

To transcribe the speech from a multimedia file that contains audio and video, you must separate the audio data from the video data. You can use the FFmpeg utility to separate the audio from the video source. For more information, see ffmpeg.org.

Tips for improving speech recognition

The following tips can help you improve the quality of speech recognition:

- How you record audio can make a big difference in the service's results. Speech recognition can be very sensitive to input audio quality. To obtain the best possible accuracy, ensure that the input audio quality is as good as possible.
  - Use a close, speech-oriented microphone (such as a headset) whenever possible and adjust the microphone settings if necessary. The service performs best when professional microphones are used to capture audio.
  - Avoid using a system's built-in microphone. The microphones that are typically installed on mobile devices and tablets are often inadequate.
  - Ensure that speakers are close to microphones. Accuracy declines as a speaker moves farther from a microphone. At a distance of 10 feet, for example, the service struggles to produce adequate results.
- Use a sampling rate no greater than 16 kHz (for broadband models) or 8 kHz (for narrowband models), and use 16 bits per sample. The service converts audio recorded at a sampling rate that is higher than the target model (16 kHz or 8 kHz) to the rate of the model. So larger frequencies do not result in enhanced recognition accuracy, but they do increase the size of the audio stream.
- Encode your audio in a format that offers data compression. By encoding your data more efficiently, you can send far more audio without exceeding the 100 MB limit. Because the dynamic range of the human voice is more limited than, say, music, speech can accommodate a bit rate that is lower than other types of audio. Nonetheless, IBM® recommends that you choose an audio format and compression algorithm carefully. For more information, see Maximizing transcription accuracy.
- Speech recognition is sensitive to background noise and nuances of human speech.
  - Engine noise, working devices, street noise, and background conversations can significantly reduce recognition accuracy.
  - Regional accents and differences in pronunciation can also reduce accuracy.

  If your audio has these characteristics, consider using acoustic model customization to improve the accuracy of speech recognition. For more information, see Understanding customization.
- To learn more about the characteristics of your audio, consider using audio metrics with your speech recognition request. If you are knowledgeable about audio-signal processing, the metrics can provide meaningful and detailed insight into your audio characteristics. For more information, see Audio metrics.

Recognizing speech with the service

Making a speech recognition request

To request speech recognition with the IBM Watson® Speech to Text service, you need to provide only the audio that is to be transcribed. The service offers the same basic transcription capabilities with each of its interfaces: the WebSocket interface, the synchronous HTTP interface, and the asynchronous HTTP interface.

The examples that follow show basic transcription requests, with no optional parameters, for each of the service's interfaces:

- The examples submit a brief FLAC file named audio-file.flac.
- The examples use the default language model, `en-US_BroadbandModel`. For more information, see Using the default model.

Understanding speech recognition results describes the service's response for these examples.

Usage requirements

Consider the following basic usage requirements when you make a speech recognition request:

- Method names are case-sensitive.
- HTTP request headers are case-insensitive.
- HTTP and WebSocket query parameters are case-sensitive.
- JSON field names are case-sensitive.
- All JSON response content is in the UTF-8 character set.
- Braces (`{ }`) are used in the documentation to indicate variable values. Omit the braces when supplying variable values.

Also consider the following service-specific requirements:

- You need to specify only the input audio. All other parameters are optional.
- If necessary, make sure to specify the `model` parameter to indicate a model that is appropriate for your language and audio.
- If you specify an invalid query parameter or JSON field as part of the input, the response includes a `warnings` field that describes the invalid argument. The request succeeds despite any warnings.

Sending audio with a request

The audio that you pass to the service must be in one of the service's supported formats. For most audio, the service can automatically detect the format. For some audio, you must specify the format with the `Content-Type` or equivalent parameter. For more information, see Audio formats. (For clarity, the following examples specify the audio format with all requests.)

With the WebSocket and synchronous HTTP interfaces, you can pass a maximum of 100 MB of audio data with a single request. With the asynchronous HTTP interface, you can pass a maximum of 1 GB of audio data. You must send at least 100 bytes of audio with any request.

If you are recognizing large amounts of audio, you can manually divide the audio into smaller chunks. But it is usually more efficient and convenient to convert the audio to a compressed, lossy format. Compression can maximize the amount of data that you can send with a single request. Especially if the audio is in WAV or FLAC format, converting it to a lossy format can make an appreciable difference.

- For more information about audio formats that use compression, see Audio formats.
- For more information about the effects of compression and about converting your audio to a format that uses it, see Data limits and compression and Audio conversion.
- For more information about transcribing the audio from a multimedia file that contains both audio and video, see Transcribing speech from video files.

Using the WebSocket interface

The WebSocket interface offers an efficient implementation that provides low latency and high throughput over a full-duplex connection. All requests and responses are sent over the same WebSocket connection.

To use the WebSocket interface, you first use the `/v1/recognize` method to establish a connection with the service. You specify parameters such as the language model and any custom models that are to be used for requests that are sent over the connection. You then register event listeners to handle responses from the service. To make a request, you send a JSON text message that includes the audio format and any additional parameters. You pass the audio as a binary message (blob), and then send a text message to signal the end of the audio.

The following example provides JavaScript code that establishes a connection and sends the text and binary messages for a recognition request. The basic example does not include the code to define all of the necessary event handlers for the connection.

```
var access_token = {access_token};
var wsURI = '{ws_url}/v1/recognize'
  + '?access_token=' + access_token;
var websocket = new WebSocket(wsURI);

websocket.onopen = function(evt) { onOpen(evt) };

function onOpen(evt) {
  var message = {
    action: 'start',
    content-type: 'audio/flac'
  };
  websocket.send(JSON.stringify(message));
  websocket.send(blob);
  websocket.send(JSON.stringify({action: 'stop'}));
}
```

Using the synchronous HTTP interface

The synchronous HTTP interface provides the simplest way to make a recognition request. You use the `POST /v1/recognize` method to make a request to the service. You pass the audio and all parameters with the single request. The following `curl` example shows a basic HTTP recognition request:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognize"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognize"
```

Using the asynchronous HTTP interface

The asynchronous HTTP interface provides a non-blocking interface for transcribing audio. You can use the interface with or without first registering a callback URL with the service. With a callback URL, the service sends callback notifications with job status and recognition results. The interface uses HMAC-SHA1 signatures based on a user-specified secret to provide authentication and data integrity for its notifications. Without a callback URL, you must poll the service for job status and results. With either approach, you use the `POST /v1/recognitions` method to make a recognition request.

The following `curl` example shows a simple asynchronous HTTP recognition request. The request does not include a callback URL, so you must poll the service to get the job status and the resulting transcript.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognitions"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognitions"
```

Understanding speech recognition results

Regardless of the interface that you use, the IBM Watson® Speech to Text service returns transcription results that reflect the parameters that you specify.

The service returns all JSON response content in the UTF-8 character set.

Basic transcription response

The service returns the following response for the examples in [Making a speech recognition request](). The examples pass only an audio file and its content type. The audio speaks a single sentence with no noticeable pauses between words.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.96,
          "transcript": "several tornadoes touch down as a line of severe thunderstorms swept through Colorado on Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

The service returns a `SpeechRecognitionResults` object, which is the top-level response object. For these simple requests, the object includes one `results` field and and one `result_index` field:

- The `results` field provides an array of information about the transcription results. For this example, the `alternatives` field includes the `transcript` and the service's `confidence` in the results. The `final` field has a value of `true` to indicate that these results will not change.
- The `result_index` field provides a unique identifier for the results. The example shows final results for a request with a single audio file that has no pauses, and the request includes no additional parameters. So the service returns a single `result_index` field with a value of `0`, which is always the initial index.

If the input audio is more complex or the request includes additional parameters, the results can contain much more information.

The alternatives field

The `alternatives` field provides an array of transcription results. For this request, the array includes just one element.

- The `transcript` field provides the results of the transcription.
- The `confidence` field is a score that indicates the service's confidence in the transcript, which for this example exceeds 90 percent.

The `final` and `result_index` fields qualify the meaning of these fields.

⚠ **Important:** Internal changes and improvements to the service can affect transcripts and confidence scores. For example, speech recognition may be improved to return more precise transcription results. Similarly, transcript and word confidence scores might change slightly as a result of improved speech recognition. Such changes are expected to be modest, but do not expect transcripts and confidence scores to remain unchanged over time.

The final field

The `final` field indicates whether the transcript shows final transcription results:

- The field is `true` for final results, which are guaranteed not to change. The service sends no further updates for final results.
- The field is `false` for interim results, which are subject to change. If you use the `interim_results` parameter with the WebSocket interface, the service returns evolving hypotheses in the form of multiple `results` fields as it transcribes the audio. For interim results, the `final` field is always `false` and the `confidence` field is always omitted.

For more information about using the WebSocket interface to obtain interim results with large speech models, previous- and next-generation models, see the following topics:

- [Interim results]()
- [Requesting interim results and low latency]()
- [How the service sends recognition results]()

The result_index field

The `result_index` field provides an identifier for the results that is unique for that request. If you request interim results, the service sends multiple `results` fields for evolving hypotheses of the input audio. The indexes for interim results for the same audio always have the same value, as do the final results for the same audio.

The same index can also be used for multiple final results of a single request. Regardless of whether you request interim results, the service can return multiple final results with the same index if your audio includes pauses or extended periods of silence. For more information, see [Pauses and silence](#).

Once you receive final results for any audio, the service sends no further results with that index for the remainder of the request. The index for any further results is incremented by one.

If your audio produces multiple final results, concatenate the `transcript` elements of the final results to assemble the complete transcription of the audio. Assemble the results in the order in which you receive them. When you assemble a complete final transcript, you can ignore interim results for which the `final` field is `false`.

Additional response content

Many speech recognition parameters impact the contents of the service's response. Some parameters can cause the service to return multiple transcription results:

- `end_of_phrase_silence_time`
- `interim_results`
- `split_transcript_at_phrase_end`

Some parameters can modify the contents of a transcript:

- `profanity_filter`
- `redaction`
- `smart_formatting`

Other parameters can add more information to the results:

- `audio_metrics`
- `keywords` and `keywords_threshold`
- `max_alternatives`
- `processing_metrics` and `processing_metrics_interval`
- `speaker_labels`
- `timestamps`
- `word_alternatives_threshold`
- `word_confidence`

For more information about the available parameters, see [Using speech recognition parameters](#) and the [Parameter summary](#).

Pauses and silence

How the service returns results depends on the interface and the model that you use for speech recognition, as well as the audio that you pass to the service. By default, the service transcribes an entire audio stream as a single utterance and returns a single final result for all of the audio. However, the service can return multiple final results in response to the following conditions:

- The audio contains a pause or extended silence between spoken words or phrases. For most languages, the default pause interval that the service uses to determine separate final results is 0.8 seconds. For Chinese, the default interval is 0.6 seconds. You can use the `end_of_phrase_silence_time` parameter to change the duration of the pause interval. For more information, see [End of phrase silence time](#).
- *For previous-generation models,* the utterance reaches a two-minute maximum. The service splits a transcript into multiple final results after two minutes of continuous processing.

The following examples show responses with two final results from the HTTP and WebSocket interfaces. The same input audio is used in both cases. The audio speaks the phrase "one two three four five six," with a one-second pause between the words "three" and "four." The examples use the default pause interval for speech recognition.

- *For the HTTP interfaces,* the service always sends a single `SpeechRecognitionResults` object. The `alternatives` array has a separate element for each final result. The response has a single `result_index` field with a value of `0`.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "one two three "
        }
      ],
      "final": true
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "four five six "
        }
      ],
      "final": true
    }
  ]
}
```

- *For the WebSocket interface,* the service sends the same results as the previous example. The response includes a single `SpeechRecognitionResults` object, the `alternatives` array has a separate element for each final result, and the response has a single `result_index` field with a value of `0`.

  ```
  {
    "result_index": 0,
    "results": [
      {
        "alternatives": [
          {
            "confidence": 0.99,
            "transcript": "one two three "
          }
        ],
        "final": true
      },
      {
        "alternatives": [
          {
            "confidence": 0.99,
            "transcript": "four five six "
          }
        ],
        "final": true
      }
    ]
  }
  ```

  With the WebSocket interface, responses for interim results contain more JSON objects. For more information about using the WebSocket interface to obtain interim results with large speech models, previous- and next-generation models, see the following topics:

  - Interim results
  - Requesting interim results and low latency
  - How the service sends recognition results

Note: Silence of 30 seconds in streamed audio can result in an inactivity timeout. For more information, see Timeouts.

Speech hesitations and hesitation markers

Speech often includes hesitations or verbal pauses, which are also referred to as disfluencies. Hesitations occur when the user inserts fillers such as "uhm", "uh", "hmm", and related non-lexical utterances while speaking. The service handles hesitations differently for large speech models, previous- and next-generation models.

Hesitations for previous-generation models

For previous-generation models, the service includes hesitation markers in transcription results for most languages. Different languages can use different hesitation markers or not indicate hesitations at all:

- *For US English,* hesitation markers are indicated by the token `%HESITATION`. Words that generate hesitation markers are `aah`, `ah`, `hm`, `hmm`, `huh`, `huh-uh`, `hum`, `ohh`, `ugh`, `uh`, `uh-huh`, `uh-oh`, `uh-uh`, `um`, and `um-hum`.
- *For Japanese,* hesitation markers typically begin with `D_`.
- *For Spanish,* the service does not produce hesitation markers.

The following example shows the token `%HESITATION` for a US English transcript:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": ". . . that %HESITATION that's a . . ."
        }
      ],
      "final": true
    }
  ]
}
```

Hesitation markers can appear in both interim and final results. Enabling smart formatting prevents hesitation markers from appearing in final results for US English. For more information, see Smart formatting.

Hesitation markers can also appear in other fields of a transcript. For example, if you request Word timestamps for the individual words of a transcript, the service reports the start and end time of each hesitation marker.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "timestamps": [
            . . .
            [
              "that",
              7.31,
              7.69
            ],
            [
              "%HESITATION",
              7.69,
              7.98
            ],
            [
              "that's",
              7.98,
              8.41
            ],
            [
              "a",
              8.41,
              8.48
            ],
            . . .
          ],
          "confidence": 0.99,
          "transcript": ". . . that %HESITATION that's a . . ."
        }
      ],
      "final": true
    }
  ]
}
```

Note: Unless you need to use them for your application, you can safely filter hesitation markers from a transcript.

Hesitations for next-generation models

For next-generation models, the service includes the actual hesitations words in all transcription results. Next-generation models treat hesitations as words, so hesitations can appear in interim results, final results, and other fields such as the results for word timestamps. Next-generation models do not produce hesitation markers, and enabling smart formatting does not cause hesitations to be removed from final results. Different languages can identify different hesitation words:

- *For US English,* common hesitation words, as for previous-generation models, are `aah`, `ah`, `hm`, `hmm`, `huh`, `huh-uh`, `hum`, `ohh`, `ugh`, `uh`, `uh-huh`, `uh-oh`, `uh-uh`, `um`, and `um-hum`. Not all of these hesitation words might appear in transcripts.
- *For Japanese,* hesitation words typically consist of half-width characters such as `ｱ`, `ｱﾉｰ`, `ｳｰﾝ`, `ｴｰﾄ`, `ﾏ`, `ﾃ`, `ﾏ`, and `ﾝｰﾄ`. Some hesitations might be recognized as full-width characters.

To increase the likelihood of seeing hesitations in your response, you can use a custom language model. In the custom model, add corpora that include the hesitations or create custom words whose sounds-likes capture the way users say the disfluencies. For more information about custom language models, see Creating a custom language model.

The following example shows the hesitation "uhm" in a US English transcript:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": ". . . that uhm that's a . . ."
        }
      ],
      "final": true
    }
  ]
}
```

Capitalization

For most languages, the service does not use capitalization in response transcripts. If capitalization is important for your application, you must capitalize the first word of each sentence and any other terms for which capitalization is appropriate.

The service applies automatic capitalization only to the following languages and models. The service always applies this capitalization, regardless of whether you use smart formatting.

- *For US English previous-generation models,* the service capitalizes many proper nouns. For example, the service returns the following transcript for the phrase *barack obama graduated from columbia university* :

  ```
  Barack Obama graduated from Columbia University
  ```

  The service does not capitalize proper nouns for US English with next-generation models.

- *For German next-generation models,* the service capitalizes many nouns. For example, the service returns the following transcript for the phrase *er braucht erst einen neuen eintrag ins vokabular punkt*:

  ```
  er braucht erst einen neuen Eintrag ins Vokabular Punkt
  ```

  The service does not capitalize nouns for German with previous-generation models.

Punctuation

The service does not insert punctuation in response transcripts by default. You must add any punctuation that you need to the service's results.

For some languages, you can use smart formatting to direct the service to substitute punctuation symbols, such as commas, periods, question marks, and exclamation points, for certain keyword strings. For more information, see Smart formatting.

The WebSocket interface

The WebSocket interface of the IBM Watson® Speech to Text service is the most natural way for a client to interact with the service. To use the WebSocket interface for speech recognition, you first use the `/v1/recognize` method to establish a persistent connection with the service. You then send text and binary messages over the connection to initiate and manage recognition requests.

Because of their advantages, WebSockets are the preferred mechanism for speech recognition. For more information, see Advantages of the WebSocket interface. For more information about the WebSocket interface and its parameters, see the API & SDK reference.

Managing a WebSocket connection

The WebSocket recognition request and response cycle has the following steps:

1. Open a connection
2. Initiate a recognition request
3. Send audio and receive recognition results
4. End a recognition request
5. Send additional requests and modify request parameters
6. Keep a connection alive
7. Close a connection

When the client sends data to the service, it *must* pass all JSON messages as text messages and all audio data as binary messages.

Note: The snippets of example code that follow are written in JavaScript and are based on the HTML5 WebSocket API. For more information about the WebSocket protocol, see the Internet Engineering Task Force (IETF) Request for Comment (RFC) 6455.

Open a connection

The Speech to Text service uses the WebSocket Secure (WSS) protocol to make the `/v1/recognize` method available at the following endpoint:

```
wss://api.{location}.speech-to-text.watson.cloud.ibm.com/instances/{instance_id}/v1/recognize
```

where `{location}` indicates where your application is hosted:

- `us-south` for Dallas
- `us-east` for Washington, DC
- `eu-de` for Frankfurt
- `au-syd` for Sydney
- `jp-tok` for Tokyo
- `eu-gb` for London
- `kr-seo` for Seoul

And `{instance_id}` is the unique identifier of your service instance.

Note: The examples in the documentation abbreviate `wss://api.{location}.speech-to-text.watson.cloud.ibm.com/instances/{instance_id}` to `{ws_url}`. All WebSocket examples call the method as `{ws_url}/v1/recognize`.

A WebSocket client calls the `/v1/recognize` method with the following query parameters to establish an authenticated connection with the service. You can specify these aspects of the request only as query parameters of the WebSocket URL.

`access_token` (*required* string)

Pass a valid access token to establish an authenticated connection with the service. You must establish the connection before the access token expires. You pass an access token only to establish an authenticated connection. Once you establish a connection, you can keep it alive indefinitely. You remain authenticated for as long as you keep the connection open. You do not need to refresh the access token for an active connection that lasts beyond the token's expiration time. Once a connection is established, it can remain active even after the token or its credentials are deleted.

- IBM Cloud Pass an Identity and Access Management (IAM) access token to authenticate with the service. You pass an IAM access token instead of passing an API key with the call. For more information, see Authenticating to IBM Cloud.
- IBM Cloud Pak for Data Pass an access token as you would with the `Authorization` header of an HTTP request. For more information, see Authenticating to IBM Cloud Pak for Data.

`model` (*optional* string)

Specifies the language model to be used for transcription. If you do not specify a model, the service uses `en-US_BroadbandModel` by default. For more information, see

- Large speech languages and models
- Previous-generation languages and models
- Next-generation languages and models
- Using the default model

`language_customization_id` (*optional* string)

Specifies the Globally Unique Identifier (GUID) of a custom language model that is to be used for all requests that are sent over the connection. The base model of the custom language model must match the value of the `model` parameter. If you include a custom language model ID, you must make the request with credentials for the instance of the service that owns the custom model. By default, no custom language model is used. For more information, see Using a custom language model for speech recognition.

`acoustic_customization_id` (*optional* string)

Specifies the GUID of a custom acoustic model that is to be used for all requests that are sent over the connection. The base model of the custom acoustic model must match the value of the `model` parameter. If you include a custom acoustic model ID, you must make the request with credentials for the instance of the service that owns the custom model. By default, no custom acoustic model is used. For more information, see Using a custom acoustic model for speech recognition.

`base_model_version` (*optional* string)

Specifies the version of the base `model` that is to be used for all requests that are sent over the connection. The parameter is intended primarily for use with custom models that are upgraded for a new base model. The default value depends on whether the parameter is used with or without a custom model. For more information, see Making speech recognition requests with upgraded custom models.

`x-watson-metadata` (*optional* string)

Associates a customer ID with all data that is passed over the connection. The parameter accepts the argument `customer_id={id}`, where `id` is a random or generic string that is to be associated with the data. You must URL-encode the argument to the parameter, for example, `customer_id%3dmy_customer_ID`. By default, no customer ID is associated with the data. For more information, see Information security.

`x-watson-learning-opt-out` (*optional* boolean)

IBM Cloud Indicates whether the service logs requests and results that are sent over the connection. To prevent IBM from accessing your data for general service improvements, specify `true` for the parameter. For more information, see Request logging.

The following snippet of JavaScript code opens a connection with the service. The call to the `/v1/recognize` method passes the `access_token` and `model` query parameters, the latter to direct the service to use the Spanish broadband model. After it establishes the connection, the client defines the event listeners (`onOpen`, `onClose`, and so on) to respond to events from the service. The client can use the connection for multiple recognition requests.

```
var access_token = '{access_token}';
var wsURI = '{ws_url}/v1/recognize'
  + '?access_token=' + access_token
  + '&model=es-ES_BroadbandModel';
var websocket = new WebSocket(wsURI);

websocket.onopen = function(evt) { onOpen(evt) };
websocket.onclose = function(evt) { onClose(evt) };
websocket.onmessage = function(evt) { onMessage(evt) };
websocket.onerror = function(evt) { onError(evt) };
```

The client can open multiple concurrent WebSocket connections to the service. The number of concurrent connections is limited only by the capacity of the service, which generally poses no problems for users.

Initiate a recognition request

To initiate a recognition request, the client sends a JSON text message to the service over the established connection. The client must send this message before it sends any audio for transcription. The message must include the `action` parameter but can usually omit the `content-type` parameter:

`action` (*required* string)

Specifies the action to be performed:

    - `start` begins a recognition request. It can also specify new parameters for subsequent requests. For more information, see [Send additional requests and modify request parameters](#).
    - `stop` signals that all audio for a request has been sent. For more information, see [End a recognition request](#).

`content-type` (*optional* string)

    Identifies the format (MIME type) of the audio data for the request. The parameter is required for the `audio/alaw`, `audio/basic`, `audio/l16`, and `audio/mulaw` formats. For more information, see [Audio formats](#).

The message can also include optional parameters to specify other aspects of how the request is to be processed and the information that is to be returned. These additional parameters include the `interim_results` parameter, which is available only with the WebSocket interface.

- For more information about all speech recognition features, see the [Parameter summary](#).
- For more information about the `interim_results` parameter, see [Interim results](#).

The following snippet of JavaScript code sends initialization parameters for the recognition request over the WebSocket connection. The calls are included in the client's `onOpen` function to ensure that they are sent only after the connection is established.

```
function onOpen(evt) {
  var message = {
    action: 'start',
    content-type: 'audio/l16;rate=22050'
  };
  websocket.send(JSON.stringify(message));
}
```

If it receives the request successfully, the service returns the following text message to indicate that it is `listening`. The `listening` state indicates that the service instance is configured (the JSON `start` message was valid) and is ready to accept audio for a recognition request.

```
{'state': 'listening'}
```

If the client specifies an invalid query parameter or JSON field for the recognition request, the service's JSON response includes a `warnings` field. The field describes each invalid argument. The request succeeds despite the warnings.

Send audio and receive recognition results

After it sends the initial `start` message, the client can begin to send audio data to the service. The client does not need to wait for the service to respond to the `start` message with the `listening` message. Once it begins listening, the service processes any audio that was sent before the `listening` message.

The client must send the audio as binary data. The client can send a maximum of 100 MB of audio data per `send` request. It must send at least 100 bytes of audio for any request. The client can send multiple requests over a single WebSocket connection. For information about using compression to maximize the amount of audio that you can pass to the service with a request, see [Data limits and compression](#).

The WebSocket interface imposes a maximum frame size of 4 MB. The client can set the maximum frame size to less than 4 MB. If it is not practical to set the frame size, the client can set the maximum message size to less than 4 MB and send the audio data as a sequence of messages. For more information about WebSocket frames, see [IETF RFC 6455](#).

How the service sends recognition results over a WebSocket connection depends on whether the client requests interim results. For more information, see [How the service sends recognition results](#).

The following snippet of JavaScript code sends audio data to the service as a binary message (blob):

```
websocket.send(blob);
```

The following snippet receives recognition hypotheses that the service returns asynchronously. The results are handled by the client's `onMessage` function.

```
function onMessage(evt) {
  console.log(evt.data);
}
```

Your code must be prepared to handle return return codes from the service. For more information, see [WebSocket return codes](#).

End a recognition request

When it is done sending the audio data for a request to the service, the client *must* signal the end of the binary audio transmission to the service in one of the following ways:

- By sending a JSON text message with the `action` parameter set to the value `stop`:

  ```
  {action: 'stop'}
  ```

- By sending an empty binary message, one in which the specified blob is empty:

  ```
  websocket.send(blob)
  ```

If the client fails to signal that the transmission is complete, the connection can time out without the service sending final results. To receive final results between multiple recognition requests, the client must signal the end of transmission for the previous request before it sends a subsequent request. After it returns the final results for the first request, the service returns another `{"state":"listening"}` message to the client. This message indicates that the service is ready to receive another request.

Send additional requests and modify request parameters

While the WebSocket connection remains active, the client can continue to use the connection to send further recognition requests with new audio. By default, the service continues to use the parameters that were sent with the previous `start` message for all subsequent requests that are sent over the same connection.

To change the parameters for subsequent requests, the client can send another `start` message with the new parameters after it receives the final recognition results and a new `{"state":"listening"}` message from the service. The client can change any parameters except for those parameters that are specified when the connection is opened (`model`, `language_customization_id`, and so on).

The following example sends a `start` message with new parameters for subsequent recognition requests that are sent over the connection. The message specifies the same `content-type` as the previous example, but it directs the service to return confidence measures and timestamps for the words of the transcription.

```
var message = {
  action: 'start',
  content-type: 'audio/l16;rate=22050',
  word_confidence: true,
  timestamps: true
};
websocket.send(JSON.stringify(message));
```

Keep a connection alive

The service terminates the session and closes the connection if an inactivity or session timeout occurs:

- An *inactivity timeout* occurs if audio is being sent by the client but the service detects no speech. The inactivity timeout is 30 seconds by default. You can use the `inactivity_timeout` parameter to specify a different value, including `-1` to set the timeout to infinity. For more information, see [Inactivity timeout](#).
- A *session timeout* occurs if the service receives no data from the client or sends no interim results for 30 seconds. You cannot change the length of this timeout, but you can extend the session by sending the service any audio data, including just silence, before the timeout occurs. You must also set the `inactivity_timeout` to `-1`. You are charged for the duration of any data that you send to the service, including the silence that you send to extend a session. For more information, see [Session timeout](#).

WebSocket clients and servers can also exchange *ping-pong frames* to avoid read timeouts by periodically exchanging small amounts of data. Many WebSocket stacks exchange ping-pong frames, but some do not. To determine whether your implementation uses ping-pong frames, check its list of features. You cannot programmatically determine or manage ping-pong frames.

If your WebSocket stack does not implement ping-pong frames and your are sending long audio files, your connection can experience a read timeout. To avoid such timeouts, continuously stream audio to the service or request interim results from the service. Either approach can ensure that the lack of ping-pong frames does not cause your connection to close.

For more information about ping-pong frames, see [Section 5.5.2 Ping](#) and [Section 5.5.3 Pong](#) of IETF RFC 6455.

Close a connection

When the client is done interacting with the service, it can close the WebSocket connection. Once the connection is closed, the client can no longer use it to send requests or to receive results. Close the connection only after the client receives all results for a request. The connection eventually times out and closes if the client does not explicitly close it.

The following snippet of JavaScript code closes an open connection:

```
websocket.close();
```

How the service sends recognition results

How the service sends speech recognition results to the client depends on whether the client requests interim results. In the JSON response to a request, final results are labeled `"final": true`, and interim results are labeled `"final": false`. For more information, see Interim results.

In the following examples, the results show the service's response for the same input audio submitted both with and without interim results. The audio speaks the phrase "one two ...*pause*... three four," with a one-second pause between the words "two" and "three." The pause is long enough to represent separate utterances. An utterance is a component of the input audio that elicits a response, usually as a result of extended silence. For more information, see Understanding speech recognition results.

If your results include multiple final results, concatenate the `transcript` elements of the final results to assemble the complete transcription of the audio. For more information, see The result_index field.

Example request without interim results

The client disables interim results by setting the `interim_results` parameter to `false` or by omitting the parameter from a request (the default argument for the parameter is `false`). The client receives a single JSON object in response only after it sends a `stop` message.

The response object can contain multiple final results for separate utterances of the audio. The service does not send the single response object until it receives a `stop` message to indicate that audio transmission for the request is complete. The structure and format of the service's response is the same regardless of whether you use a previous- or next-generation model.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "one two "
        }
      ],
      "final": true
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "three four "
        }
      ],
      "final": true
    }
  ]
}
```

Example request with interim results

The client requests interim results as follows:

- *For previous-generation models,* by setting the `interim_results` parameter to `true`.
- *For next-generation models,* by setting both the `interim_results` and `low_latency` parameters to `true`. To receive interim results with a next-generation model, the model must support low latency and both the `interim_results` and `low_latency` parameters must be set to `true`.
    - For more information about which next-generation models support low latency, see Supported next-generation models.
    - For more information about the interaction of the `interim_results` and `low_latency` parameters with next-generation models, including examples that demonstrate the different combinations of parameters, see Requesting interim results and low latency.

The client receives multiple JSON objects in response. The service returns separate response objects for each interim result and for each each final result that is generated by the audio. The service sends at least one interim result for each final result.

The service sends responses as soon as they are available. It does not wait for a `stop` message to send its results, though the `stop` message is still required to signal the end of transmission for the request. The structure and format of the service's response is the same regardless of whether you use a previous- or next-generation model.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "transcript": "one "
        }
      ],
      "final": false
    }
  ]
}{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "transcript": "one two "
        }
      ],
      "final": false
    }
  ]
}{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "one two "
        }
      ],
      "final": true
    }
  ]
}{
  "result_index": 1,
  "results": [
    {
      "alternatives": [
        {
          "transcript": "three "
        }
      ],
      "final": false
    }
  ]
}{
  "result_index": 1,
  "results": [
    {
      "alternatives": [
        {
          "transcript": "three four "
        }
      ],
      "final": false
    }
  ]
}{
  "result_index": 1,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "three four "
        }
      ],
      "final": true
    }
  ]
}
```

Example WebSocket exchanges

The following examples show a series of exchanges between a client and the Speech to Text service over a single WebSocket connection. The examples focus on the exchange of messages and data. They do not show opening and closing the connection. (The examples are based on a previous-generation model, so the final transcript for each response includes a `confidence` field.)

First example exchange

In the first exchange, the client sends audio that contains the string `Name the Mayflower`. The client sends a binary message with a single chunk of PCM (`audio/l16`) audio data, for which it indicates the required sampling rate. The client does not wait for the `{"state":"listening"}` response from the service to begin sending the audio data and to signal the end of the request. Sending the data immediately reduces latency because the audio is available to the service as soon as it is ready to handle a recognition request.

- The client sends:

```
{
  "action": "start",
  "content-type": "audio/l16;rate=22050"
}
<binary audio data>
{
  "action": "stop"
}
```

- The service responds:

```
{"state": "listening"}
{"results": [{"alternatives": [{"transcript": "name the mayflower ",
             "confidence": 0.91}], "final": true}], "result_index": 0}
{"state":"listening"}
```

Second example exchange

In the second exchange, the client sends audio that contains the string `Second audio transcript`. The client sends the audio in a single binary message and uses the same parameters that it specified in the first request.

- The client sends:

```
<binary audio data>
{
  "action": "stop"
}
```

- The service responds:

```
{"results": [{"alternatives": [{"transcript": "second audio transcript ",
             "confidence": 0.99}], "final": true}], "result_index": 0}
{"state":"listening"}
```

Third example exchange

In the third exchange, the client again sends audio that contains the string `Name the Mayflower`. It sends a binary message with a single chunk of PCM audio data. But this time, the client sends a new `start` message that requests interim results from the service.

- The client sends:

```
{
  "action": "start",
  "content-type": "audio/l16;rate=22050",
  "interim_results": true
}
<binary audio data>
```

```
{
  "action": "stop"
}
```

- The service responds:

```
{"results": [{"alternatives": [{"transcript": "name "}],
             "final": false}], "result_index": 0}
{"results": [{"alternatives": [{"transcript": "name may "}],
             "final": false}], "result_index": 0}
{"results": [{"alternatives": [{"transcript": "name may flour "}],
             "final": false}], "result_index": 0}
{"results": [{"alternatives": [{"transcript": "name the mayflower ",
             "confidence": 0.91}], "final": true}], "result_index": 0}
{"state":"listening"}
```

WebSocket return codes

The service can send the following return codes to the client over the WebSocket connection:

- `1000` indicates normal closure of the connection, meaning that the purpose for which the connection was established has been fulfilled.
- `1002` indicates that the service is closing the connection due to a protocol error.
- `1006` indicates that the connection closed abnormally.
- `1009` indicates that the frame size exceeded the 4 MB limit.
- `1011` indicates that the service is terminating the connection because it encountered an unexpected condition that prevents it from fulfilling the request.

If the socket closes with an error, the client receives an informative message of the form `{"error":"{message}"}` before the socket closes. Use the `onerror` event handler to respond appropriately. For more information about WebSocket return codes, see IETF RFC 6455.

Note: The WebSocket implementations of the SDKs can return different or additional response codes. For more information, see the API & SDK reference.

The synchronous HTTP interface

The synchronous HTTP interface of the IBM Watson® Speech to Text service provides a single `POST /v1/recognize` method for requesting speech recognition with the service. This method is the simplest means of obtaining a transcript. It offers two ways of submitting a speech recognition request:

- The first sends all of the audio in a single stream via the body of the request. You specify the parameters of the operation as request headers and query parameters. For more information, see Making a basic HTTP request.
- The second sends the audio as a multipart request. You specify the parameters of the request as a combination of request headers, query parameters, and JSON metadata. For more information, see Making a multipart HTTP request.

Submit a maximum of 100 MB and a minimum of 100 bytes of audio data with a single request. For information about audio formats and about using compression to maximize the amount of audio that you can send with a request, see Supported audio formats. For information about all methods of the HTTP interface, see the API & SDK reference.

Making a basic HTTP speech recognition request

The HTTP `POST /v1/recognize` method provides a simple means of transcribing audio. You pass all audio via the body of the request and specify the parameters as request headers and query parameters.

The method returns results only after it processes all of the audio for a request. The method is appropriate for batch processing but not for live speech recognition. Use the WebSocket interface to transcribe live audio.

If your data consists of multiple audio files, the recommended means of submitting the audio is by sending multiple requests, one for each audio file. You can submit the requests in a loop, optionally with parallelism to improve performance. You can also use multipart speech recognition to pass multiple audio files with a single request.

Basic request example

The following example sends a recognition request for a single FLAC file named `audio-file.flac`. The request omits the `model` query parameter to use the default language model, `en-US_BroadbandModel`. For more information, see Using the default model.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize"
```

The example returns the following transcript for the audio:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.96,
          "transcript": "several tornadoes touch down as a line of severe thunderstorms swept through Colorado on Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

Making a multipart HTTP speech recognition request

Note: The asynchronous HTTP interface, WebSocket interface, and Watson SDKs do not support multipart speech recognition.

The `POST /v1/recognize` method also supports multipart requests for speech recognition. You pass all audio data as multipart form data. You specify some parameters as request headers and query parameters, but you pass JSON metadata as form data to control most aspects of the transcription.

Multipart speech recognition is intended for the following use cases:

- To pass multiple audio files with a single speech recognition request.
- With browsers for which JavaScript is disabled. Multipart requests based on form data do not require the use of JavaScript.
- When the parameters of a recognition request are greater than 8 KB, which is the limit imposed by most HTTP servers and proxies. For example, spotting a very large number of keywords can increase the size of a request beyond this limit. Multipart requests use form data to avoid this constraint.

The following sections describe the parameters that you use for multipart requests and show an example request.

Parameters for multipart requests

You specify a number of parameters as form data, request headers, or query parameters. For more information about request headers and query parameters, see the Parameter summary.

Form data

You specify the following parameters of multipart speech recognition requests as form data:

`metadata` (*required* object)

A JSON object that provides the transcription parameters for the request. The object must be the first part of the form data. The information describes the audio in the subsequent parts of the form data. See JSON metadata for multipart requests.

`upload` (*required* file)

One or more audio files as the remainder of the form data for the request. All audio files must have the same format. With the `curl` command, include a separate `--form` option for each file of the request.

Request headers

You specify the following parameters as request headers:

`Content-Type` (*required* string)

Specify `multipart/form-data` to indicate how data is passed to the method. You specify the content type of the audio with the JSON `part_content_type` parameter.

`Transfer-Encoding` (*optional* string)

Specify `chunked` to stream the audio data to the service. Omit the parameter if you send all audio with a single request.

Query parameters

You specify the following parameters as query parameters:

`model` (*optional* string)

The identifier of the model that is to be used with the request. The default is `en-US_BroadbandModel`. For more information, see Using the default model.

`language_customization_id` (*optional* string)

The GUID of a custom language model that is to be used with the request.

`acoustic_customization_id` (*optional* string)

The GUID of a custom acoustic model that is to be used with the request.

`base_model_version` (*optional* string)

The version of the specified base model that is to be used with the request.

JSON metadata for multipart requests

The JSON metadata that you pass with a multipart request can include the following fields:

- `part_content_type` (string)
- `data_parts_count` (integer)
- `customization_weight` (number)
- `inactivity_timeout` (integer)
- `keywords` (string[])
- `keywords_threshold` (number)
- `max_alternatives` (integer)
- `word_alternatives_threshold` (number)
- `word_confidence` (boolean)
- `timestamps` (boolean)
- `profanity_filter` (boolean)
- `smart_formatting` (boolean)
- `speaker_labels` (boolean)
- `grammar_name` (string)
- `redaction` (boolean)
- `end_of_phrase_silence_time` (double)
- `split_transcript_at_phrase_end` (boolean)
- `speech_detector_sensitivity` (number)
- `background_audio_suppression` (number)

- `low_latency` (boolean)
- `character_insertion_bias` (float)

Only the following two parameters are specific to multipart requests:

- The `part_content_type` field is *optional* for most audio formats. It is required for the `audio/alaw`, `audio/basic`, `audio/l16`, and `audio/mulaw` formats. It specifies the format of the audio in the following parts of the request. All audio files must be in the same format.
- The `data_parts_count` field is *optional* for all requests. It specifies the number of audio files that are sent with the request. The service applies end-of-stream detection to the last (and possibly the only) data part. If you omit the parameter, the service determines the number of parts from the request.

All other parameters of the metadata are optional. For descriptions of all available parameters, see the Parameter summary.

Multipart request example

The following example shows how to pass a multipart recognition request with the `POST /v1/recognize` method. The request passes two audio files, **audio-file1.flac** and **audio-file2.flac**. The `metadata` parameter provides most parameters of the request; the `upload` parameters provide the audio files.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: multipart/form-data" \
--form metadata="{\"part_content_type\":\"application/octet-stream\", \
  \"data_parts_count\":2, \
  \"timestamps\":true, \
  \"word_alternatives_threshold\":0.9, \
  \"keywords\":[\"colorado\",\"tornado\",\"tornadoes\"], \
  \"keywords_threshold\":0.5}" \
--form upload="@{path}audio-file1.flac" \
--form upload="@{path}audio-file2.flac" \
"{url}/v1/recognize"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: multipart/form-data" \
--form metadata="{\"part_content_type\":\"application/octet-stream\", \
  \"data_parts_count\":2, \
  \"timestamps\":true, \
  \"word_alternatives_threshold\":0.9, \
  \"keywords\":[\"colorado\",\"tornado\",\"tornadoes\"], \
  \"keywords_threshold\":0.5}" \
--form upload="@{path}audio-file1.flac" \
--form upload="@{path}audio-file2.flac" \
"{url}/v1/recognize"
```

The example returns the following transcript for the audio files. The service returns the results for the two files in the order in which they are sent. (The example output abbreviates the results for the second file.)

```
{
  "result_index": 0,
  "results": [
    {
      "word_alternatives": [
        {
          "start_time": 0.03,
          "alternatives": [
            {
              "confidence": 0.96,
              "word": "the"
            }
          ],
          "end_time": 0.09
        },
        {
          "start_time": 0.09,
          "alternatives": [
          {
            "confidence": 0.96,
            "word": "latest"
          }
        ],
        "end_time": 0.62
      },
      {
        "start_time": 0.62,
        "alternatives": [
          {
            "confidence": 0.96,
            "word": "weather"
          }
        ],
        "end_time": 0.87
      },
      {
        "start_time": 0.87,
        "alternatives": [
          {
            "confidence": 0.96,
            "word": "report"
          }
        ],
        "end_time": 1.5
      }
    ],
    "keywords_result": {},
    "alternatives": [
      {
        "timestamps": [
          [
            "the",
            0.03,
            0.09
          ],
          [
            "latest",
            0.09,
            0.62
          ],
          [
            "weather",
            0.62,
            0.87
          ],
          [
            "report",
            0.87,
            1.5
          ]
        ],
        "confidence": 0.99,
        "transcript": "the latest weather report "
      }
    ],
    "final": true
  },
  {
    "word_alternatives": [
      {
        "start_time": 0.15,
        "alternatives": [
          {
            "confidence": 1.0,
            "word": "a"
          }
        ],
        "end_time": 0.3
      },
    {
      "start_time": 0.3,
      "alternatives": [
        {
          "confidence": 1.0,
          "word": "line"
        }
      ],
      "end_time": 0.64
    },
    . . .
    {
      "start_time": 4.58,
      "alternatives": [
        {
          "confidence": 0.98,
          "word": "Colorado"
        }
      ],
      "end_time": 5.16
    },
    {
      "start_time": 5.16,
      "alternatives": [
        {
          "confidence": 0.98,
          "word": "on"
        }
      ],
      "end_time": 5.32
    },
    {
      "start_time": 5.32,
      "alternatives": [
        {
          "confidence": 0.98,
          "word": "Sunday"
        }
      ],
      "end_time": 6.04
    }
  ],
  "keywords_result": {
    "tornadoes": [
      {
        "normalized_text": "tornadoes",
        "start_time": 3.03,
        "confidence": 0.98,
        "end_time": 3.84
      }
    ],
    "colorado": [
      {
        "normalized_text": "Colorado",
        "start_time": 4.58,
        "confidence": 0.98,
        "end_time": 5.16
      }
    ]
  },
  "alternatives": [
    {
      "timestamps": [
        [
          "a",
          0.15,
          0.3
        ],
        [
          "line",
          0.3,
              0.64
            ],
            . . .
            [
              "Colorado",
              4.58,
              5.16
            ],
            [
              "on",
              5.16,
              5.32
            ],
            [
              "Sunday",
              5.32,
              6.04
            ]
          ],
          "confidence": 0.99,
          "transcript": "a line of severe thunderstorms with several
possible tornadoes is approaching Colorado on Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

The asynchronous HTTP interface

The asynchronous HTTP interface of the IBM Watson® Speech to Text service provides methods for transcribing audio via non-blocking calls to the service. The interface employs user-specified secret strings and digital signatures to provide a level of security for requests that are made over the HTTP protocol. To use the asynchronous interface, you can

- Register a callback URL to be notified by the service of the job status and the results automatically.
- Poll the service to obtain the job status and the results manually.

The two approaches are not mutually exclusive. You can elect to receive callback notifications but still poll the service for the latest status or contact the service to retrieve results manually. The following sections describe how to use the asynchronous HTTP interface with either approach.

Submit a maximum of 1 GB and a minimum of 100 bytes of audio data with a single request. For information about audio formats and about using compression to maximize the amount of audio that you can send with a request, see Supported audio formats. For more information about the individual methods of the interface, see the API & SDK reference.

Note: The asynchronous HTTP interface does not support multipart speech recognition. You can use only the synchronous HTTP interface for multipart requests. For more information, see Making a multipart HTTP speech recognition request .

Usage models

When you work with the service's asynchronous HTTP interface, you can elect to learn about job status and receive results in the following ways:

- By using callback notifications:
  1. Call the `POST /v1/register_callback` method to register a callback URL with the service. You can provide an optional user-specified secret string to enable authentication and data integrity for callbacks that are sent to the URL.
  2. Call the `POST /v1/recognitions` method with an already registered callback URL to which the service sends notifications when the status of the job changes. You specify a list of events of which to be notified. By default, the service sends notifications when a job is started, when it is complete, and if an error occurs. You can also request the results of the request in the completion notification. Otherwise, you need to use the `GET /v1/recognitions/{id}` method to retrieve the results.
- By polling the service:
  1. Call the `POST /v1/recognitions` method without a callback URL, events, or a user token.
  2. Periodically call the `GET /v1/recognitions` method to check the status of the most recent jobs or the `GET /v1/recognitions/{id}` method to check the status of a specific job.
  3. If you check job status with the `GET /v1/recognitions` method, call the `GET /v1/recognitions/{id}` method to retrieve the results of the job once it is complete.

The two approaches can be used together. You can still poll the service to obtain the latest status for a job that is created with a callback URL. For example, you might want to obtain the status of a job if notifications are taking a while to arrive. You might also check the status if you suspect that you missed one or more notifications due to a service or network error.

Registering a callback URL

You register a callback URL by calling the `POST /v1/register_callback` method. Once you register a callback URL, you can use it to receive notifications for an indefinite number of jobs. The registration process comprises four steps:

1. You call the `POST /v1/register_callback` method and pass a callback URL. Optionally, you can also specify a user-specified secret. The service uses the secret to compute keyed-hash message authentication code (HMAC) Secure Hash Algorithm 1 (SHA1) signatures for authentication and data integrity. The following example registers a user callback that responds at the URL `http://{user_callback_path}/results`. The call includes a user secret of `ThisIsMySecret`.

   IBM Cloud

   ```
   $ curl -X POST -u "apikey:{apikey}" \
   "{url}/v1/register_callback?callback_url=http://{user_callback_path}/results&user_secret=ThisIsMySecret"
   ```

   IBM Cloud Pak for Data

   ```
   $ curl -X POST \
   --header "Authorization: Bearer {token}" \
   "{url}/v1/register_callback?callback_url=http://{user_callback_path}/results&user_secret=ThisIsMySecret"
   ```

2. The service attempts to validate, or allowlist, the callback URL if it is not already registered by sending a `GET` request to the callback URL. The service passes a random alphanumeric challenge string via the `challenge_string` query parameter of the request. The request includes an `Accept` header that specifies `text/plain` as the required response type.

   If the call to the `register_callback` method included a user secret, the `GET` request from the service also includes an `X-Callback-Signature` header that specifies the HMAC-SHA1 signature of the challenge string. The service calculates the signature by using the user secret as the key.

   ```
   GET http://{user_callback_path}/results?challenge_string=n9ArPGMQ36Hiu7QC
   header: X-Callback-Signature {HMAC-SHA1_signature}
   ```

3. You respond to the `GET` request from the service with status code 200. Include the challenge string that was sent by the service in the response. Include the string in plain text in the body of the response, and set the `Content-Type` response header to `text/plain`.

   If the initial `POST` request included a user secret, you can calculate an HMAC-SHA1 signature of the challenge string by using the secret as the key. If the `GET` request was sent by the service, the signature matches the value specified by the `X-Callback-Signature` header.

   ```
   response code: 200 OK
   body: n9ArPGMQ36Hiu7QC
   ```

4. The service checks whether the challenge string is returned in the body of the response to its `GET` request. If it is, the service allowlists the callback URL and responds to your original `POST` request with status code 201. The body of the response includes a JSON object with a `status` field that has the value `created` and a `url` field that has the value of your callback URL.

   ```
   response code: 201 Created
   body: {
     "status": "created",
     "url": "http://{user_callback_path}/results"
   }
   ```

The service sends only a single `GET` request to a callback URL during the registration process. The service must receive a reply with response code 200 that includes the challenge string in its body within five seconds. Otherwise, it does not allowlist the URL. Instead, it sends status code 400 in response to the `POST /v1/register_callback` request. If the callback URL was already successfully allowlisted, the service sends status code 200 in response to the initial `POST` request.

Security considerations

When you successfully use the `POST /v1/register_callback` method to register a callback URL, the service allowlists the URL to indicate that it is verified for use with callback notifications. If you specify a user secret with the registration call, allowlisting also means that the URL has been validated for added security. Specifying a user secret provides authentication and data integrity for requests that use the callback URL with the asynchronous HTTP interface.

The service uses the user secret to compute an HMAC-SHA1 signature over the payload of every callback notification that it sends to the URL. The service sends the signature via the `X-Callback-Signature` header with each notification. The client can use the secret to compute its own signature of each notification payload. If its signature matches the value of the `X-Callback-Signature` header, the client knows that the notification was sent by the service and that its contents have not been altered during transmission. This knowledge guarantees that the client is not the victim of a man-in-middle-attack.

HTTPS is ideal for production applications. But during application development and prototyping, the HTTP-based callback notifications that are supported by the service can simplify and accelerate the development process by avoiding the expense of HTTPS.

Unregistering a callback URL

You can unregister an allowlisted callback URL at any time by calling the `POST /v1/unregister_callback` method. Unregistering a callback URL can be useful for testing your application with the service. Once you unregister a callback URL, you can no longer use it with asynchronous recognition requests.

Unregister a callback URL example

The following example unregisters a previously registered callback URL:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
"{url}/v1/unregister_callback?callback_url=http://{user_callback_path}/results"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
"{url}/v1/unregister_callback?callback_url=http://{user_callback_path}/results"
```

Creating a job

You create a recognition job by calling the `POST /v1/recognitions` method. How you learn the status and results of the job depends on the approach you use and the parameters you pass:

- *To use callback notifications*, include the `callback_url` query parameter to specify a URL to which the service is to send callback notifications when the status of the job changes. You can also specify the following optional query parameters:
  - `events` to subscribe to a list of notification events. By default, the service sends callback notifications when the job is started (the `recognitions.started` event), when the job is complete (the `recognitions.completed` event), and if an error occurs (the `recognitions.failed` event). You can specify a subset of the events or use the `recognitions.completed_with_results` event instead of the `recognitions.completed` event to include the results with the job-completed notification.
  - `user_token` to specify a string that is to be included with each notification for the job. Because you can use the same callback URL with an indefinite number of jobs, you can leverage user tokens to differentiate notifications for different jobs.
- *To use polling*, omit the `callback_url`, `events`, and `user_token` query parameters. You must then use the `GET /v1/recognitions` or `GET /v1/recognitions/{id}` methods to check the status of the job, using the latter to retrieve the results when the job is complete.

In both cases, you can include the `results_ttl` query parameter to specify the number of minutes for which the results are to remain available after the job completes. The new job is owned by the instance of the service whose credentials are used to create it.

In addition to the previous parameters, which are specific to the asynchronous interface, the `POST /v1/recognitions` method supports most of the same parameters as the WebSocket and synchronous HTTP interfaces. For more information, see the Parameter summary.

Callback notifications

If the job is created with a callback URL, the service sends an HTTP `POST` callback notification to the registered URL when a specified event occurs. The body of a basic notification consists of a JSON object with the following structure:

```
{
  "id": "{job_id}",
  "event": "{recognitions_status}",
  "user_token": "{user_token}"
}
```

The `id` field identifies the ID of the job that generated the callback, and the `event` field identifies the event that triggered the callback. The `user_token` field includes the user token for the job if one was specified. Otherwise, the field is an empty string. If the event is `recognitions.completed_with_results`, the object includes a `results` field that provides the results of the recognition request. The client can respond to the callback notification with status code 200.

If the callback URL was registered with a user secret, the service also sends the `X-Callback-Signature` header with the callback notification. The header specifies the HMAC-SHA1 signature of the body of the request. The service calculates the signature by using the user secret as a key. The client can calculate the signature of the payload for the callback notification to be sure that it matches the signature in the header. For example, the following simple Python code computes the signature on the string `notification_payload`:

```
from hashlib import sha1
import hmac
hashed = hmac.new(user_secret, notification_payload, sha1)
signature = hashed.digest().encode("base64").rstrip('\n')
```

Create a job with a callback URL example

The following example creates a job that is associated with the previously allowlisted callback URL `http://{user_callback_path}/results`. The example passes the user token `job25` to identify the job in callback notifications that are sent by the service. The call uses the default events, so the user must call the `GET /v1/recognitions/{id}` method to retrieve the results when the service sends a callback notification to indicate that the job is complete. The call sets the `timestamps` query parameter of the recognition request to `true`.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognitions?callback_url=http://{user_callback_path}/results&user_token=job25&timestamps=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognitions?callback_url=http://{user_callback_path}/results&user_token=job25&timestamps=true"
```

The service returns the status of the request, which is `waiting` to indicate that the service is preparing the job for processing. The response includes the creation time, the job ID, and the URL at which to obtain more information about the job.

```
{
  "created": "2016-08-17T19:15:17.926Z",
  "id": "4bd734c0-e575-21f3-de03-f932aa0468a0",
  "url": "{url}/v1/recognitions/4bd734c0-e575-21f3-de03-f932aa0468a0",
  "status": "waiting"
}
```

Create a job with polling example

The following example creates a job that is not associated with a callback URL. The user must poll the service to learn when the job is complete and then retrieve the results with the `GET /v1/recognitions/{id}` method. Like the previous example, the call sets the `timestamps` parameter of the recognition request to `true`.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognitions?timestamps=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognitions?timestamps=true"
```

The service returns a status of `processing` to indicate that it is already processing the job, along with the creation time, the job ID, and the URL for getting information about the job.

```
{
  "created": "2016-08-17T19:13:23.622Z",
  "id": "4bb1dca0-f6b1-11e5-80bc-71fb7b058b20",
  "url": "{url}/v1/recognitions/4bb1dca0-f6b1-11e5-80bc-71fb7b058b20",
  "status": "processing"
}
```

Checking the status and retrieving the results of a job

You call the `GET /v1/recognitions/{id}` method to check the status of the job that is specified with the `id` path parameter. The response always includes the ID and status of the job and its creation and update times. If the status is `completed`, the response also includes the results of the recognition request.

The `GET /v1/recognitions/{id}` method is the only way to retrieve job results if

- The job was submitted without a callback URL.
- The job was submitted with a callback URL but without specifying the `recognitions.completed_with_results` event.
- The job is not one of the 100 latest outstanding jobs. When you omit the `id` path parameter, only the 100 latest jobs are returned.

However, you can still use the method to retrieve the results for a job that specified a callback URL and the `recognitions.completed_with_results` event. You can retrieve the results for any job as many times as you want while they remain available. A job and its results remain available until you delete them with the `DELETE /v1/recognitions/{id}` method or until the job's time to live expires, whichever comes first. By default, results expire after one week unless you specify a different time to live with the `results_ttl` parameter of the `POST /v1/recognitions` method.

Check job status without results example

The following example checks the status of the job with the specified ID:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/recognitions/{job_id}"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/recognitions/{job_id}"
```

The job is not yet complete, so the response does not include the results.

```
{
  "id": "4bb1dca0-f6b1-11e5-80bc-71fb7b058b20",
  "created": "2016-08-17T19:13:23.622Z",
  "updated": "2016-08-17T19:13:24.434Z",
  "status": "processing"
}
```

Check job status with results example

The following example requests the status of the job with the specified ID:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/recognitions/{job_id}"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/recognitions/{job_id}"
```

The job is complete, so the response includes the results of the request.

```
{
  "id": "398fcd80-330a-22ba-93ce-1a73f454dd98",
  "results": [
    {
      "result_index": 0,
      "results": [
        {
          "final": true,
          "alternatives": [
            {
              "transcript": "several tornadoes touch down as a line of severe thunderstorms swept through Colorado on Sunday ",
              "timestamps": [
                [
                  "several",
                  1,
                  1.52
                ],
                [
                  "tornadoes",
                  1.52,
                  2.15
                ],
                . . .
                [
                  "Sunday",
                  5.74,
                  6.33
                ]
              ],
              "confidence": 0.96
            }
          ]
        }
      ]
    }
  ],
  "created": "2016-08-17T19:11:04.298Z",
  "updated": "2016-08-17T19:11:16.003Z",
  "status": "completed"
}
```

Checking the status of the latest jobs

You call the `GET /v1/recognitions` method to check the status of the latest jobs. The method returns the status of the most recent 100 outstanding jobs that are associated with the credentials with which it is called. The method returns the ID and status of each job, along with its creation and update times. If a job was created with a callback URL and a user token, the method also returns the user token for the job.

The response includes one of the following states:

- `waiting` if the service is preparing the job for processing. This status is the initial state of all jobs. The job remains in this state until the service has the capacity to begin processing it.
- `processing` if the service is actively processing the job.
- `completed` if the service has finished processing the job. If the job specified a callback URL and the event `recognitions.completed_with_results`, the service sent the results with the callback notification. Otherwise, use the `GET /v1/recognitions/{id}` method to obtain the results.
- `failed` if the job failed for some reason.

A job and its results remain available until you delete them with the `DELETE /v1/recognitions/{id}` method or until the job's time to live expires, whichever comes first.

Check the status of the latest jobs example

The following example requests the status of the latest current jobs that are associated with the caller's credentials:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/recognitions"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/recognitions"
```

The user has three outstanding jobs in various states. The first job was created with a callback URL and a user token.

```
{
  "recognitions": [
    {
      "id": "4bd734c0-e575-21f3-de03-f932aa0468a0",
      "created": "2016-08-17T19:15:17.926Z",
      "updated": "2016-08-17T19:15:17.926Z",
      "status": "waiting",
      "user_token": "job25"
    },
    {
      "id": "4bb1dca0-f6b1-11e5-80bc-71fb7b058b20",
      "created": "2016-08-17T19:13:23.622Z",
      "updated": "2016-08-17T19:13:24.434Z",
      "status": "processing"
    },
    {
      "id": "398fcd80-330a-22ba-93ce-1a73f454dd98",
      "created": "2016-08-17T19:11:04.298Z",
      "updated": "2016-08-17T19:11:16.003Z",
      "status": "completed"
    }
  ]
}
```

Deleting a job

You can use the `DELETE /v1/recognitions/{id}` method to delete the job that is specified with the `id` path parameter. You typically delete a job after you obtain its results from the service. Once you delete a job, its results are no longer available. You cannot delete a job that the service is actively processing.

By default, the service maintains the results of each job until the job's time to live expires. The default time to live is one week, but you can use the `results_ttl` parameter of the `POST /v1/recognitions` method to specify the number of minutes that the service is to maintain the results.

Delete a job example

The following example deletes the job with the specified ID:

IBM Cloud

```
$ curl -X DELETE -u "apikey:{apikey}" \
"{url}/v1/recognitions/{job_id}"
```

IBM Cloud Pak for Data

```
$ curl -X DELETE \
--header "Authorization: Bearer {token}" \
"{url}/v1/recognitions/{job_id}"
```

Using speech recognition parameters

Audio transmission and timeouts

The IBM Watson® Speech to Text service lets you pass audio to the service all at once or stream audio to the service. For audio streaming, the service enforces timeouts to ensure ongoing session activity.

Audio transmission

With the WebSocket interface, audio data is always streamed to the service over the connection. You can pass data through the socket all at once, or you can pass data for the live-use case as it becomes available. The service returns results as they become available.

With the HTTP interfaces, you can transmit audio to the service in either of the following ways:

- *One-shot delivery.* You omit the `Transfer-Encoding` request header and pass all of the audio data to the service at one time as a single delivery.
- *Streaming.* You set the `Transfer-Encoding` request header to the value `chunked` and stream the data over a persistent connection. The data does not need to exist fully before you stream it to the service. You can stream the data as it becomes available. The service sends results only when it receives the final chunk, which you indicate by sending an empty chunk.

  For more information about streaming chunked audio with the `Transfer-Encoding` header, see

  - Chunked transfer encoding
  - Transfer Codings in *IETF RFC 7320 HTTP/1.1: Message Syntax and Routing*

With the HTTP interfaces, the service always transcribes the entire audio stream before sending any results. The results can include multiple `transcript` elements to indicate phrases that are separated by pauses. Concatenate the `transcript` elements to assemble the complete transcript.

The service enforces timeouts on a streaming session. It can terminate a streaming session if it detects an extended period of silence or receives no audio during a 30-second period. For more information about timeouts and how to avoid them, see Timeouts.

Audio transmission example

The following example request specifies `chunked` for the `Transfer-Encoding` header to use streaming mode. The connection remains open to accept additional chunks of audio.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--header "Transfer-Encoding: chunked" \
--data-binary @{path}audio-file1.flac \
"{url}/v1/recognize"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--header "Transfer-Encoding: chunked" \
--data-binary @{path}audio-file1.flac \
"{url}/v1/recognize"
```

Timeouts

When you initiate a streaming session with the HTTP or WebSocket speech recognition methods, the service enforces inactivity and session timeouts. If a timeout lapses during a streaming session, the service closes the connection. Your application must recover gracefully from possible closed connections.

When you stream audio over HTTP, the service sends a space character in its response every 20 seconds. The service does this to improve usability by avoiding the 30-second HTTP REST inactivity timeout. To keep the connection alive while recognition is ongoing, the service continues to send this space character until it completes its transcription. The space character has no effect on JSON-encoded response data.

Note: This HTTP inactivity timeout is different from the service's inactivity timeout. The WebSocket interface is not subject to this HTTP timeout.

Inactivity timeout

An *inactivity timeout* (HTTP status code 400) occurs when the service is receiving audio but detects only continuous silence or non-speech activity (no speech) for 30 seconds. The service sends the error message `No speech detected for 30s`. The inactivity timeout is useful, for example, for terminating a session when a user simply walks away from a live microphone.

The default inactivity timeout is 30 seconds. You can override this value by using the `inactivity_timeout` parameter. Specify a larger value to increase the inactivity timeout. Specify a value of `-1` to set the inactivity timeout to infinity. You are charged for all audio that you send to the service, including silence, so increasing the inactivity timeout can incur additional charges for a streaming session that sends only silence.

Inactivity timeout example

The following example request sets the inactivity timeout to 60 seconds. The request sends an initial file to begin the streaming session.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Transfer-Encoding: chunked" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file1.flac \
"{url}/v1/recognize?inactivity_timeout=60"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Transfer-Encoding: chunked" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file1.flac \
"{url}/v1/recognize?inactivity_timeout=60"
```

Session timeout

A *session timeout* (HTTP status code 408) occurs when you fail to send sufficient audio to keep a streaming session active. The service can consider a session idle and trigger a session timeout for the following reasons:

- You fail to send at least 15 seconds of audio to the service in any 30-second window.

  Until you send the last chunk to indicate the end of the stream, you must send at least 15 seconds of audio within any 30-second period. The audio can be silence if you set the `inactivity_timeout` parameter to a larger value or to `-1`. You are charged for the duration of any audio that you send to the service, including silence.

- You stream audio at a rate that is much slower than real-time.

  Ideally, you would initiate a request to establish a session just before you obtain audio for transcription. You would then maintain the session by sending audio at a rate that is close to real-time.

You do not need to worry about the session timeout after you send the last chunk to indicate the end of the stream. The service continues to process the audio until it returns the final transcription results.

When you transcribe a long audio stream, the service can take more than 30 seconds to process the audio and generate a response. The service does not begin to calculate the session timeout until it finishes processing all audio that it has received. The service's processing time cannot cause the session to exceed the 30-second session timeout.

For example, if you send one hour of audio in the first 10 seconds of a session, the service might take 300 seconds to process the audio. To keep this session alive, you would need to send at least 15 more seconds of some audio, including silence, no later than 340 seconds into the session.

In this example, if you were to send another 15 seconds of audio at the 100-second mark of the session, the service might spend an additional two seconds processing this audio. In this case, you would need to send 15 more seconds of audio no later 342 seconds into the session.

⚠ **Important:** Do not rely on processing time or on whether you have received results to determine whether a streaming session is idle. Assume that the service can process all audio instantly, and send data to the service accordingly. If you stream audio in real-time, do not fall behind in sending audio at one-half real-time (15 seconds of audio) in any 30-second window. This rate is typically sufficient to accommodate network latency and delays.

Interim results and low latency

With the WebSocket interface, the IBM Watson® Speech to Text service supports interim results, which are intermediate transcription hypotheses that arrive before final results. For the WebSocket and HTTP interfaces, most next-generation models also offer low latency to return results even more quickly than they already do, though transcription accuracy might be reduced.

When you use next-generation models that support low latency with the WebSocket interface, you must enable both interim results and low latency to get interim results. Only next-generation models that support low latency can return interim results. Note that previous- and next-generation models return slightly different results in certain cases.

Interim results

Note: The interim results feature is available only with the WebSocket interface. The parameter is not available with large speech models.

Interim results are intermediate transcription hypotheses that are likely to change before the service returns its final results. The service returns interim results as soon as it generates them. Interim results are useful for interactive applications and real-time transcription, and for long audio streams, which can take a while to transcribe.

Interim results evolve as the service's processing of an utterance progresses. They arrive more often and more quickly than final results. You can use them to enable your application to respond more quickly or to gauge the progress of transcription. Once its processing of an utterance is complete, the service sends final results that represent its best transcription of the audio for that utterance.

- *Interim results* are identified in a transcript with the field `"final": false`. The service can update interim results with more accurate transcriptions as it processes further audio. The service delivers one or more interim results for each final result.
- *Final results* are identified with the field `"final": true`. The service makes no further updates to final results.

How you request interim results depends on the type of model that you are using:

- *For a previous-generation model,* set the `interim_results` parameter to `true` in the JSON `start` message. Interim results are available for all previous-generation models.
- *For a next-generation model,* set the `interim_results` and `low_latency` parameters to `true` in the JSON `start` message. Interim results are available only for next-generation models that support low latency, and only if both interim results and low latency are enabled. For more information, see Requesting interim results and low latency.

To disable interim results for any model, omit the `interim_results` parameter or set it to `false`. Disable interim results if you are doing offline or batch transcription.

Interim results example

The following abbreviated WebSocket example requests interim results. The service sends multiple response objects. It sets the `final` attribute to `true` only for the final results. The request implicitly uses the default previous-generation `en-US_BroadbandModel`.

```
var access_token = {access_token};
var wsURI = '{ws_url}/v1/recognize'
  + '?access_token=' + access_token;
var websocket = new WebSocket(wsURI);

websocket.onopen = function(evt) { onOpen(evt) };
function onOpen(evt) {
  var message = {
    action: 'start',
    content-type: 'audio/l16;rate=22050',
    interim_results: true
  };
  websocket.send(JSON.stringify(message));
  websocket.send(blob);
}

websocket.onmessage = function(evt) { onMessage(evt) };
function onMessage(evt) {
  console.log(evt.data);
}
```

The response includes a single utterance with no pauses.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "transcript": "several to "
        }
      ],
      "final": false
    }
  ]
}{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "transcript": "several tornadoes "
        }
      ],
      "final": false
    }
  ]
}{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "transcript": "several tornadoes swept through "
        }
      ],
      "final": false
    }
  ]
}{
  . . .
}{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.96,
          "transcript": "several tornadoes swept through Colorado on Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

Low latency

Note: The `low_latency` parameter is available for most next-generation models. The parameter is not available with large speech models and previous-generation models.

The next-generation multimedia and telephony models have generally faster response times than the previous-generation models. But there might be cases where you want to receive results even more quickly. With next-generation models that support low latency, you can set the `low_latency` parameter to `true` to receive results more quickly. For more information about the next-generation models that support low latency, see Supported next-generation language models.

With low latency, the service achieves faster results at the possible expense of transcription accuracy. When low latency is enabled, the service segments the audio into smaller chunks to optimize for speed over accuracy. This trade-off might be acceptable if your application needs lower response time more than it does the highest possible accuracy. For example, low latency is ideal for use cases such as closed captioning, conversational applications, and live customer service in the voice channel of the IBM® watsonx™ Assistant service.

Omitting the `low_latency` parameter can produce more accurate results and is the recommended approach for most use cases. The `low_latency` parameter is `false` by default.

The nature of the response to a request that includes low latency depends on the interface that is used:

- With the HTTP interfaces, the service waits for all of the audio to arrive before sending a response. It then sends a single stream of bytes in response. The response can include multiple transcription elements with multiple final results, and it can be sent incrementally. But it is a single stream of data.
- With the WebSocket interface, the service sends final results as they become available. It can send multiple independent responses in the form of different streams of bytes. The connection is bidirectional and full duplex, and requests and responses can continue to flow back and forth over a single connection for as long as it remains active.

Low-latency restrictions

The low-latency feature has the following usage restrictions:

- *Low latency is available only for some next-generation models.* For next-generation models that do not support low-latency, if you include the `low_latency` parameter with a request, the service fails with status code 400:

  ```
  $ {
    "code": 400,
    "code_description": "Bad Request",
    "error": "low_latency is not a supported feature for model {model_id}"
  }
  ```

- *Low latency is not available for previous-generation models.* For previous-generation models, if you include the `low_latency` parameter with a request, the service generates a warning:

  ```
  "warnings": [
    "Unknown arguments: low_latency."
  ]
  ```

Low-latency example

The following synchronous HTTP example requests low latency with the `en-US_Telephony` model. The example sets the `low_latency` query parameter to `true`.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?model=en-US_Telephony&low_latency=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?model=en-US_Telephony&low_latency=true"
```

Requesting interim results and low latency

Note: To receive interim results with a next-generation model, the model must support low latency and both the `interim_results` and `low_latency` parameters must be set to `true`. The service does not support interim results for next-generation models that do not support low latency.

When you use next-generation models that support low latency, you can request both interim results and low latency with the WebSocket interface. However, the `interim_results` parameter behaves differently when it is used with next-generation models.

The following table describes the interaction between the `interim_results` and `low_latency` parameters and the results that you receive depending on their settings. The default values for both the `interim_results` and `low_latency` parameters are `false`.

`interim_results`	`low_latency`	**Results**
`interim_results=false`	`low_latency=false`	The service sends only final results. It returns a single JSON object that includes results for all utterances when transcription is complete. See [Example 1: Interim results and low latency are both false]().

`interim_results=false`	`low_latency=true`	The service sends only final results. It returns a single JSON object that includes results for all utterances when transcription is complete. But because `low_latency` is `true`, the service returns the final results more quickly. See Example 2: Interim results is false and low latency is true.
`interim_results=true`	`low_latency=false`	The service sends only final results. It returns multiple JSON objects for individual utterances as it performs transcription. The advantage of setting `interim_results` to `true` is that results for utterances arrive as they become complete. You do not need to wait for all utterances to be transcribed. See Example 3: Interim results is true and low latency is false .
`interim_results=true`	`low_latency=true`	The service returns interim results as it develops transcription hypotheses, and it sends final results for utterances as they become complete. The service delivers one or more interim results for each final result. The quality of interim results is identical to what you receive with previous-generation models. But because `low_latency` is `true`, the service returns interim and final results more quickly. See Example 4: Interim results and low latency are both true .

Table 1. Interaction of interim_results and low_latency parameters

Interim results and low-latency examples

The following abbreviated WebSocket code shows how to request interim results, low-latency results, or both with the WebSocket interface. It specifies the following parameters:

- The `model` query parameter of the `/v1/recognize` request passes the next-generation `en-US_Telephony` model, which supports low latency.
- The `inactivity_timeout` parameter of the JSON `start` message sets the inactivity timeout to `-1` (infinity), which prevents the request from timing out.
- Both the `interim_results` and `low_latency` parameters are specified with the JSON `start` message of the request.

To show examples of results with all possible combinations of the parameters, the arguments for `interim_results` and `low_latency` are set to `true` or `false` in the examples in the following sections.

```
var access_token = {access_token};
var wsURI = '{ws_url}/v1/recognize'
  + '?access_token=' + access_token
  + '&model=en-US_Telephony';
var websocket = new WebSocket(wsURI);

websocket.onopen = function(evt) { onOpen(evt) };
function onOpen(evt) {
  var message = {
    action: 'start',
    content-type: 'audio/wav',
    inactivity_timeout: -1,
    interim_results: {true | false},
    low_latency: {true | false}
  };
  websocket.send(JSON.stringify(message));
  websocket.send(blob);
}

websocket.onmessage = function(evt) { onMessage(evt) };
function onMessage(evt) {
  console.log(evt.data);
}
```

The WAV file that is passed to the service includes a single sentence with two embedded multi-second pauses: "Thunderstorms could produce ... *pause*... large hail ...*pause*... and heavy rain." The pauses are long enough to represent separate utterances and thus generate multiple final results. Because each response includes multiple final results, one per utterance, you must assemble the final results into a single string to see the full transcript.

Example 1: Interim results and low latency are both false

This example sets both `interim_results` and `low_latency` to `false`. The service returns only final results as a single JSON object.

```
  var message = {
    action: 'start',
    content-type: 'audio/wav',
    inactivity_timeout: -1,
    interim_results: false,
    low_latency: false
  };
```

```
{
  "result_index": 0,
  "results": [
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "thunderstorms could produce ",
          "confidence": 0.94
        }
      ]
    },
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "large hail ",
          "confidence": 0.91
        }
      ]
    },
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "and heavy rain ",
          "confidence": 0.86
        }
      ]
    }
  ]
}
```

Example 2: Interim results is false and low latency is true

This example sets `interim_results` to `false` and `low_latency` to `true`. The service returns only final results as a single JSON object.

```
  var message = {
    action: 'start',
    content-type: 'audio/wav',
    inactivity_timeout: -1,
    interim_results: false,
    low_latency: true
  };
```

```
{
  "result_index": 0,
  "results": [
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "thunderstorms could produce ",
          "confidence": 0.94
        }
      ]
    },
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "large hail ",
          "confidence": 0.91
        }
      ]
    },
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "and heavy rain ",
          "confidence": 0.86
        }
      ]
    }
  ]
}
```

Example 3: Interim results is true and low latency is false

This example sets `interim_results` to `true` and `low_latency` to `false`. The service returns only final results, but it returns each result as a separate JSON object.

```
  var message = {
    action: 'start',
    content-type: 'audio/wav',
    inactivity_timeout: -1,
    interim_results: true,
    low_latency: false
  };
```

```
{
  "result_index": 0,
  "results": [
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "thunderstorms could produce ",
          "confidence": 0.94
        }
      ]
    }
  ]
}{
  "result_index": 1,
  "results": [
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "large hail ",
          "confidence": 0.91
        }
      ]
    }
  ]
}{
  "result_index": 2,
  "results": [
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "and heavy rain ",
          "confidence": 0.86
        }
      ]
    }
  ]
}
```

Example 4: Interim results and low latency are both true

This example sets both `interim_results` and `low_latency` to `true`. The service returns both interim and final results, and it returns each results as a separate JSON object.

```
  var message = {
    action: 'start',
    content-type: 'audio/wav',
    inactivity_timeout: -1,
    interim_results: true,
    low_latency: true
  };
```

```
{
  "result_index": 0,
  "results": [
    {
      "final": false,
      "alternatives": [
        {
          "transcript": "th "
        }
      ]
    }
  ]
}{
  "result_index": 0,
  "results": [
    {
      "final": false,
      "alternatives": [
        {
          "transcript": "thunderstorms "
        }
      ]
    }
  ]
}{
  "result_index": 0,
  "results": [
    {
      "final": false,
      "alternatives": [
        {
          "transcript": "thunderstorms could produc "
        }
      ]
    }
  ]
}{
  "result_index": 0,
  "results": [
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "thunderstorms could produce ",
          "confidence": 0.94
        }
      ]
    }
  ]
}{
  "result_index": 1,
  "results": [
    {
      "final": false,
      "alternatives": [
        {
          "transcript": "large "
        }
      ]
    }
  ]
}{
  "result_index": 1,
  "results": [
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "large hail ",
          "confidence": 0.91
        }
      ]
    }
  ]
}{
  "result_index": 2,
  "results": [
    {
      "final": false,
      "alternatives": [
        {
          "transcript": "and hea "
        }
      ]
    }
  ]
}{
  "result_index": 2,
  "results": [
    {
      "final": true,
      "alternatives": [
        {
          "transcript": "and heavy rain ",
          "confidence": 0.86
        }
      ]
    }
  ]
}
```

Speech activity detection

The IBM Watson® Speech to Text service offers two speech activity detection parameters to control what audio is used for speech recognition. The parameters specify the service's sensitivity to non-speech events and to background noise. The parameters are independent: You can use them individually or together.

Note: Speech activity detection is supported for most language models. For more information, see [Language model support](#).

How speech activity detection works

Speech activity detection consumes the input audio stream and determines which parts of the stream to pass for speech recognition. Speech recognition is adversely affected by background speech and noise, causing the service to transcribe the wrong words, to produce words where none are present, or to omit words that are part of the input audio. The speech activity detection feature can help ensure that only relevant audio is processed for speech recognition.

You can use the feature to control the following aspects of speech recognition:

- *Suppress background speech.* Call-center data often contains cross-talk ("overhearing") from other agents. You can set a volume threshold below which such background speech is ignored.
- *Suppress background noise.* Some audio, such as speech recorded in a factory, can contain a high level of background noise. You can set a threshold below which such background noise is ignored.
- *Suppress non-speech audio events.* Background music and tone events, such as audio played to a client who is waiting on hold on a telephone line, can cause inaccurate recognition. Silence can also result in unnecessary recognition or transcription errors. You can set a threshold below which such events are ignored.

By default, speech activity detection is configured to provide optimal performance for the general case for each model. For specific cases, the default settings might not be optimal and can lead either to slow transcription or to word insertions and deletions. You are encouraged to experiment with different settings to determine which values work best for your audio.

Speech detector sensitivity

Use the `speech_detector_sensitivity` parameter to adjust the sensitivity of speech activity detection. Use the parameter to suppress word insertions from music, coughing, and other non-speech events. The service biases the audio it passes for speech recognition by evaluating chunks of the input audio against prior models of speech and non-speech activity.

Specify a float value between 0.0 and 1.0. The default value is 0.5, which provides a reasonable compromise for the level of sensitivity. A value of 0.0 suppresses all audio (no speech is transcribed). A value of 1.0 suppresses no audio (speech detection sensitivity is disabled). The values increase on a monotonic curve of sensitivity versus speech. Specifying one or two decimal places of precision (for example, `0.55`) is typically more than sufficient.

This parameter can affect both the quality and the latency of speech recognition:

- Lower values can decrease latency because less audio is potentially passed for speech recognition. However, a low setting might discard chunks of audio that contain actual speech, losing viable content from the transcript.
- Higher values can increase latency because more audio is potentially passed for speech recognition. However, a high setting might pass chunks of audio that contain non-speech events, adding spurious content to the transcript.

Speech detector sensitivity example

The following example request specifies a value of 0.6 for the `speech_detector_sensitivity` parameter with the synchronous HTTP interface. The service recognizes slightly more potential non-speech events than it would by default.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file1.flac \
"{url}/v1/recognize?speech_detector_sensitivity=0.6"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file1.flac \
"{url}/v1/recognize?speech_detector_sensitivity=0.6"
```

Background audio suppression

Use the `background_audio_suppression` parameter to suppress background audio based on its volume to prevent it from being transcribed as speech. Use the parameter to suppress side conversations or background noise. For example, use this parameter when there is a relatively steady and quiet (low signal energy) background sound. Because such noise can interfere with transcription, it can produce content where no actual speech occurs in the audio.

Specify a float value in the range of 0.0 to 1.0. The default value is 0.0, which provides no suppression (background audio suppression is disabled). A value of 0.5 provides a reasonable level of audio suppression for general usage. A value of 1.0 suppresses all audio (no speech is transcribed). The values increase on a monotonic curve. Specifying one or two decimal places of precision (for example, `0.55`) is typically more than sufficient.

This parameter can also affect both the quality and the latency of speech recognition. However, because background noise suppression is disabled by default, setting the parameter to a value greater than zero can only improve latency. But higher values can gradually reduce the audio that is passed for speech recognition, which can cause valid content to be lost from the transcript.

Background audio suppression example

The following example request specifies a value of 0.5 for the `background_audio_suppression` parameter with the synchronous HTTP interface. The service suppresses a reasonable level of background audio.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file1.flac \
"{url}/v1/recognize?background_audio_suppression=0.5"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file1.flac \
"{url}/v1/recognize?background_audio_suppression=0.5"
```

Language model support

The `speech_detector_sensitivity` and `background_audio_suppression` parameters are supported for use with the following language models:

- *For large speech models and next-generation models,* the parameters are supported with all models.
- *For previous-generation models,* the parameters are supported with most models. The following models do *not* support speech activity detection at this time. The parameters are ignored if used with these models.
  - Arabic broadband model (`ar-MS_BroadbandModel`)
  - Brazilian Portuguese broadband model (`pt-BR_BroadbandModel`)
  - Chinese broadband model (`zh-CN_BroadbandModel`)
  - Chinese narrowband model (`zh-CN_NarrowbandModel`)
  - German broadband model (`de-DE_BroadbandModel`)

Speech audio parsing

The IBM Watson® Speech to Text service provides multiple features that determine how the service is to parse audio to produce final transcription results.

- End of phrase silence time specifies the duration of the pause interval at which the service splits a transcript into multiple final results.
- Split transcript at phrase end directs the services to split a transcript into multiple final results for semantic features such as sentences.
- Character insertion bias directs the service to favor character sequences of shorter or greater length as it develops transcription hypotheses with next-generation models.

End of phrase silence time

The `end_of_phrase_silence_time` parameter specifies the duration of the pause interval at which the service splits a transcript into multiple final results. If the service detects pauses or extended silence before it reaches the end of the audio stream, its response can include multiple final results. Silence indicates a point at which the speaker pauses between spoken words or phrases. For most languages, the default pause interval is 0.8 seconds; for Chinese the default interval is 0.6 seconds. For more information, see Paauses and silence.

By using the `end_of_phrase_silence_time` parameter, you can specify a double value between 0.0 and 120.0 seconds that indicates a different pause interval:

- A value greater than 0 specifies the interval that the service is to use for speech recognition.
- A value of 0 indicates that the service is to use the default interval. It is equivalent to omitting the parameter.

You can use the parameter to strike a balance between how often a final result is produced and the accuracy of the transcription:

- A longer pause interval produces fewer final results, and each result covers a larger segment of audio. Larger segments tend to be more accurate because the service has more context with which to transcribe the audio.

  However, larger pause intervals directly impact the latency of the final results. After the last word of the input audio, the service must wait for the indicated interval to expire before it returns its response. The service waits to ensure that the input audio does not continue beyond the longer interval. (With the WebSocket interface, setting the parameter can affect the latency and accuracy of the final results, regardless of whether you request interim results.)

- A shorter pause interval yields more final results, but because each segment of audio is smaller, transcription accuracy might not be as good. Smaller segments are acceptable when latency is crucial and transcription accuracy is not expected to degrade or any degradation is acceptable.

  Also, a multi-phrase grammar recognizes, or matches, responses only within a single final result. If you use a grammar to recognize multiple strings, you can increase the pause interval to avoid receiving multiple final results.

Increase the interval when accuracy is more important than latency. Decrease the interval when the speaker is expected to say short phrases or single-word responses.

Note: *For previous-generation models,* the service generates a final result for audio when an utterance reaches a two-minute maximum. The service splits a transcript into multiple final results after two minutes of continuous processing.

End of phrase silence time example

The following example requests show the effect of using the `end_of_phrase_silence_time` parameter. The audio speaks the phrase "one two three four five six," with a one-second pause between the words "three" and "four." The speaker might be reading a six-digit number from an identification card, for example, and pause to confirm the number.

The first example uses the default pause interval of 0.8 seconds:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize"
```

Because the pause is greater than the default interval, the service splits the transcript at the pause. The `confidence` of both results is `0.99`, so transcription accuracy is very good despite the pause.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "one two three "
        }
      ],
      "final": true
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "four five six "
        }
      ],
      "final": true
    }
  ]
}
```

But suppose speech recognition is performed with a grammar that expects the user to speak six digits in a single-phrase response. Because the service splits the transcript at the one-second pause, the results are empty. The grammar is applied to each final result, but neither result, "one two three" nor "four five six", contains six digits.

The second example uses the same audio but sets the `end_of_phrase_silence_time` to 1.5 seconds:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?end_of_phrase_silence_time=1.5"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?end_of_phrase_silence_time=1.5"
```

Because this value is greater than the length of the speaker's pause, the service returns a single final result that contains the entire spoken phrase. A grammar that expects to find six digits recognizes this transcript.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "transcript": "one two three four five six "
        }
      ],
      "final": true
    }
  ]
}
```

Split transcript at phrase end

The `split_transcript_at_phrase_end` parameter directs the service to split the transcript into multiple final results based on semantic features of the input. Setting the parameter to `true` causes the service to split the transcript at the conclusion of meaningful phrases such as sentences. The service bases its understanding of semantic features on the base language model that you use with the request along with a custom language model or grammar that you use. Custom language models and grammars can influence how and where the service splits a transcript.

If you apply a custom language model to speech recognition, the service is likely to split the transcript based on the content and nature of the model's corpora. For example, a model whose corpora include many short sentences biases the service to mimic the corpora and split the input into multiple, shorter final results. Similarly, a grammar that expects a series of six digits can influence the service to insert splits to accommodate that series of digits.

However, the degree to which custom language models and grammars influence the service's splitting of a transcript depends on many factors. Such factors include the amount of training data with which the custom model is built and the qualities of the audio itself. Small changes in the audio can affect the results.

Regardless, the service can still produce multiple final results in response to pauses and silence. If you omit the `split_transcript_at_phrase_end` parameter or set it to `false`, the service splits transcripts based solely on the pause interval. The pause interval has precedence over splitting due to semantic features. And if used together on a single request, the `end_of_phrase_silence_time` parameter has precedence over the `split_transcript_at_phrase_end` parameter. For more information, see [End of phrase silence time](#).

Note: Split transcript at phrase end is not available with large speech models.

Split transcript at phrase end results

When the split transcript at phrase end feature is enabled, each final result in the transcript includes an additional `end_of_utterance` field that identifies why the service split the transcript at that point:

- `full_stop` - A full semantic stop, such as for the conclusion of a grammatical sentence. The insertion of splits is influenced by the base language model and biased by custom language models and grammars, as described previously.
- `silence` - A pause or silence that is at least as long as the pause interval. You can use the `end_of_phrase_silence_time` parameter to control the length of the pause interval.
- `end_of_data` - The end of the input audio stream.
- `reset` - The amount of audio that is currently being processed exceeds the two-minute maximum. The service splits the transcript to avoid excessive memory use.

Split transcript at phrase end example

The following example requests demonstrate the use of the `split_transcript_at_phrase_end` parameter. The audio speaks the phrase "I have an identification card. The number is one two three four five six." The speaker pauses for one second between the words "three" and "four."

The first example shows the results for a request that omits the parameter:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize"
```

The service returns two final results, splitting the transcript only at the speaker's extended pause:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.93,
          "transcript": "I have a valid identification card the number is one two three "
        }
      ],
      "final": true
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "four five six "
        }
      ],
      "final": true
    }
  ]
}
```

The second example recognizes the same audio but sets `split_transcript_at_phrase_end` to `true`:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?split_transcript_at_phrase_end=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?split_transcript_at_phrase_end=true"
```

The service returns three final results, adding a result for the semantic stop after the first sentence. Each result includes the `end_of_utterance` field to identify the reason for the split.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.92,
          "transcript": "I have a valid identification card "
        }
      ],
      "final": true,
      "end_of_utterance": "full_stop"
    },
    {
      "alternatives": [
        {
          "confidence": 0.97,
          "transcript": "the number is one two three "
        }
      ],
      "final": true,
      "end_of_utterance": "silence"
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "transcript": "four five six "
        }
      ],
      "final": true,
      "end_of_utterance": "end_of_data"
    }
  ]
}
```

Character insertion bias

The `character_insertion_bias` parameter is a functionality that is available for all next-generation models. The parameter is not available for large speech models and previous-generation models.

The `character_insertion_bias` parameter controls the service's bias for competing strings of different lengths during speech recognition. With large speech models and next-generation models, the service parses audio character by character. As it does, it establishes hypotheses of previous character strings to help determine viable next characters. During this process, it collects candidate strings of different lengths.

By default, each model uses a default `character_insertion_bias` of 0.0. This value is optimized to produce the best balance between hypotheses with different numbers of characters. The default is typically adequate for most speech recognition. However, certain use cases might benefit from favoring hypotheses with shorter or longer strings of characters. In such cases, specifying a change from the default can improve speech recognition.

You can use the `character_insertion_bias` parameter to indicate that the service is to favor shorter or longer strings as it considers subsequent characters for its hypotheses. The value you provide depends on the characteristics of your audio. The range of acceptable values is from -1.0 to 1.0:

- *Negative values* cause the service to prefer hypotheses with shorter strings of characters.
- *Positive values* cause the service to prefer hypotheses with longer strings of characters.

As your value approaches -1.0 or 1.0, the impact of the parameter becomes more pronounced. To determine the most effective value for your scenario, start by setting the value of the parameter to a small increment, such as -0.1, -0.05, 0.05, or 0.1, and assess how the value impacts the transcription results. Then experiment with different values as necessary, adjusting the value by small increments to gauge how much to deviate from the model's default.

Character insertion bias example

The following example request directs the service to marginally favor longer strings of character hypotheses by setting the `character_insertion_bias` parameter to `0.1`. This is a small initial increment to determine the effect of the change on transcription results.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?model=en-US_Telephony&character_insertion_bias=0.1"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?model=en-US_Telephony&character_insertion_bias=0.1"
```

Speaker labels

The speaker labels feature is a functionality that is available for all languages.

With speaker labels, the IBM Watson® Speech to Text service identifies which individuals spoke which words in a multi-participant exchange. You can use the feature to create a person-by-person transcript of an audio stream. For example, you can use it to develop analytics for a call-center or meeting transcript, or to animate an exchange with a conversational robot or avatar. For best performance, use audio that is at least a minute long. (Labeling who spoke what and when is sometimes referred to as *speaker diarization*.)

Speaker labels are optimized for two-speaker scenarios. They work best for telephone conversations that involve two people in an extended exchange. They can handle up to six speakers, but more than two speakers can result in variable performance. Two-person exchanges are typically conducted over narrowband (telephony) media, but you can use speaker labels with supported narrowband and broadband (multimedia) models.

To use the feature, you set the `speaker_labels` parameter to `true` for a recognition request; the parameter is `false` by default. The service identifies speakers by individual words of the audio. It relies on a word's start and end time to identify its speaker. (If you are recognizing multichannel audio, you can instead send each channel for transcription separately. For more information about this alternative approach, see Speaker labels for multichannel audio.)

Note: Setting the `speaker_labels` parameter to `true` forces the `timestamps` parameter to be `true`, regardless of whether you disable timestamps with the request. For more information, see Word timestamps.

Speaker labels example

An example is the best way to show how speaker labels work. The following example requests speaker labels with a speech recognition request:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-multi.flac \
"{url}/v1/recognize?model=en-US&speaker_labels=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-multi.flac \
"{url}/v1/recognize?model=en-US&speaker_labels=true"
```

The response includes arrays of timestamps and speaker labels. The numeric values that are associated with each element of the `timestamps` array are the start and end times of each word in the `speaker_labels` array.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "timestamps": [
            [
              "hello",
              0.28,
              0.79
            ],
            [
              "yeah",
              1.07,
              1.53
            ],
            [
              "yeah",
              1.66,
              2.02
            ],
            [
              "how's",
              2.15,
              2.44
            ],
            [
              "Billy",
              2.45,
              3.03
            ],
            [
              "good",
              3.57,
              3.95
            ]
          ]
          "confidence": 0.82,
          "transcript": "hello yeah yeah how's Billy good "
        }
      ],
      "final": true
    }
  ],
  "speaker_labels": [
    {
      "from": 0.28,
      "to": 0.79,
      "speaker": 2,
      "confidence": 0.52,
      "final": false
    },
    {
      "from": 1.07,
      "to": 1.53,
      "speaker": 1,
      "confidence": 0.63,
      "final": false
    },
    {
      "from": 1.66,
      "to": 2.02,
      "speaker": 2,
      "confidence": 0.54,
      "final": false
    },
    {
      "from": 2.15,
      "to": 2.44,
      "speaker": 2,
      "confidence": 0.51,
      "final": false
    },
    {
      "from": 2.45,
      "to": 3.03,
      "speaker": 2,
      "confidence": 0.55,
      "final": false
    },
    {
      "from": 3.57,
      "to": 3.95,
      "speaker": 1,
      "confidence": 0.63,
      "final": true
    }
  ]
}
```

The `transcript` field shows the final transcript of the audio, which lists the words as they were spoken by all participants. By comparing and synchronizing the speaker labels with the timestamps, you can reassemble the conversation as it occurred. The `confidence` field for each speaker label indicates the service's confidence in its identification of the speaker.

Timestamps (0.00 - 2.14)	Speaker labels (0.00 - 2.14)	Timestamps (2.15 - 3.95)	Speaker labels (2.15 - 3.95)

`0.28,`
`0.79`
`"hello",`
	`"from": 0.28,`
	`"to": 0.79,`
	`"speaker": 2,`
	`"confidence": 0.52,`
	`"final": false`
		`2.15,`
		`2.44`
		`"how's",`
			`"from": 2.15,`
			`"to": 2.44,`
			`"speaker": 2,`
			`"confidence": 0.51,`
			`"final": false`

`1.07,`
`1.53`
`"yeah",`
	`"from": 1.07,`
	`"to": 1.53,`
	`"speaker": 1,`
	`"confidence": 0.63,`
	`"final": false`
		`2.45,`
		`3.03`
		`"Billy",`
			`"from": 2.45,`
			`"to": 3.03,`
			`"speaker": 2,`
			`"confidence": 0.55,`
			`"final": false`

`1.66,`
`2.02`
`"yeah",`
	`"from": 1.66,`
	`"to": 2.02,`
	`"speaker": 2,`
	`"confidence": 0.54,`
	`"final": false`
		`3.57,`
		`3.95`
		`"good",`
			`"from": 3.57,`
			`"to": 3.95,`
			`"speaker": 1,`
			`"confidence": 0.63,`
			`"final": true`

Table 1. Speaker labels example

The results clearly capture the brief two-person exchange:

- **Speaker 2** - "Hello?"
- **Speaker 1** - "Yeah?"
- **Speaker 2** - "Yeah, how's Billy?"
- **Speaker 1** - "Good."

In the example, the `final` field for each speaker label but the last is `false`. The service sets the `final` field to `true` only for the last speaker label that it returns. That value of `true` indicates that the service has completed its analysis of the speaker labels.

Speaker IDs for speaker labels

The example also illustrates an interesting aspect of speaker IDs. In the `speaker` fields, the first speaker has an ID of `2` and the second has an ID of `1`. The service develops a better understanding of the speaker patterns as it processes the audio. Therefore, it can change speaker IDs for individual words, and can also add and remove speakers, until it generates its final results.

As a result, speaker IDs might not be sequential, contiguous, or ordered. For instance, IDs typically start at `0`, but in the previous example the earliest ID is `1`. The service likely omitted an earlier word assignment for speaker `0` based on further analysis of the audio. Omissions can happen for later speakers, as well. Omissions can result in gaps in the numbering and produce results for two speakers who are labeled, for example, `0` and `2`. Another consideration is that speakers can leave a conversation. So a participant who contributes only to the early stages of a conversation might not appear in later results.

Requesting interim results for speaker labels

⚠ **Important:** When you use speaker labels with interim results, the service duplicates the final speaker label object. For more information, see Known limitations.

With the WebSocket interface, you can request interim results as well as speaker labels (for more information, see Interim results). Final results are generally better than interim results. But interim results can help identify the evolution of a transcript and the assignment of speaker labels. Interim results can indicate where transient speakers and IDs appeared or disappeared. However, the service can reuse the IDs of speakers that it initially identifies and later reconsiders and omits. Therefore, an ID might refer to two different speakers in interim and final results.

When you request both interim results and speaker labels, final results for long audio streams might arrive well after initial interim results are returned. It is also possible for some interim results to include only a `speaker_labels` field without the `results` and `result_index` fields for the transcript as a whole. If you do not request interim results, the service returns final results that include `results` and `result_index` fields and a single `speaker_labels` field.

With interim results, a `final` field with a value of `false` for a word in the `speaker_labels` field can indicate that the service is still processing the audio. The service might change its identification of the speaker or its confidence for individual words before it is done. The service sends final results when the audio stream is complete or in response to a timeout, whichever occurs first. The service sets the `final` field to `true` only for the last word of the speaker labels that it returns in either case.

Performance considerations for speaker labels

As noted previously, the speaker labels feature is optimized for two-person conversations, such as communications with a call center. Therefore, you need to consider the following potential performance issues:

- Performance for audio with a single speaker can be poor. Variations in audio quality or in the speaker's voice can cause the service to identify extra speakers who are not present. Such speakers are referred to as hallucinations.
- Similarly, performance for audio with a dominant speaker, such as a podcast, can be poor. The service tends to miss speakers who talk for shorter amounts of time, and it can also produce hallucinations.
- Performance for audio with more than six speakers is undefined. The feature can handle a maximum of six speakers.
- Performance for cross-talk, or speaker overlap, is poor. Cross-talk can be difficult or impossible to transcribe accurately for any audio.
- Performance for short utterances can be less accurate than for long utterances. The service produces better results when participants speak for longer amounts of time, at least 30 seconds per speaker. The relative amount of audio that is available for each speaker can also affect performance.
- Performance can degrade for the first 30 seconds of speech. It usually improves to a reasonable level after 1 minute of audio, as the service receives more data to work with.

As with all transcription, performance can also be affected by poor audio quality, background noise, a person's manner of speech, and other aspects of the audio. IBM continues to refine and improve the performance of the speaker labels feature. For more information about the latest improvements, see IBM Research AI Advances Speaker Diarization in Real Use Cases.

Speaker labels for multichannel audio

As described in Channels, the Speech to Text service downmixes audio with multiple channels to one-channel mono and uses that audio for speech recognition. The service performs this conversion for all audio, including audio that is passed to generate speaker labels.

If you have multichannel audio that records each speaker's contributions on a separate channel, you can use the following alternative approach to achieve the equivalent of speaker labels:

1. Extract each individual channel from the audio.
2. Transcribe the audio from each channel separately and without using the `speaker_labels` parameter. Include the `timestamps` parameter to learn precise timing information for the words of the transcripts.
3. Merge the results of the individual requests to re-create a single transcript of the complete exchange. You can use the timestamps of the individual requests to reconstruct the conversation.

The multichannel approach has some advantages over the use of the `speaker_labels` parameter:

- Because it relies on basic speech recognition, the service can transcribe audio that is in any supported language. Speakers labels are restricted to a subset of the available languages.
- Because each speaker has a dedicated channel, the service can provide more accurate results than it can for speaker labels. The service does not need to determine which speaker is talking.
- Because each speaker has a dedicated channel, the service can accurately transcribe cross-talk, or speaker overlap. On a merged channel, cross-talk can be difficult or impossible to recognize accurately.

But the approach also has some disadvantages that you need to be aware of:

- You must separate the channels prior to sending the audio and then merge the resulting transcripts to reconstruct the conversation. The use of timestamps greatly simplifies the reconstruction process.
- IBM Cloud You are charged for all audio that you send to the service, including silence. If you send the audio from all channels in its entirety, you effectively multiply the amount of audio being recognized by the number of channels. For example, if you submit separate requests for both channels of a two-channel exchange that lasts for five minutes, your requests require ten minutes of audio processing. If your pricing plan uses per-minute pricing, this doubles the cost of speech recognition. You can consider preprocessing the audio to eliminate silence, but that approach makes it much more difficult to merge the results of the separate requests.

Keyword spotting and word alternatives

The IBM Watson® Speech to Text service can identify user-specified keywords in its transcription results. It can also suggest alternative words that are acoustically similar to the words of a transcript. In both cases, the keywords and word alternatives must meet a user-specified level of confidence.

Keyword spotting

Note: The `keywords` and `keywords_threshold` parameters are supported only with previous-generation models, not with next-generation models.

The keyword spotting feature detects specified strings in a transcript. The service can spot the same keyword multiple times and report each occurrence. The service spots keywords only in the final results, not in interim results. By default, the service does no keyword spotting.

To use keyword spotting, you must specify both of the following parameters:

- Use the `keywords` parameter to specify an array of strings to be spotted. The service spots no keywords if you omit the parameter or specify an empty array. A keyword string can include more than one token. For example, the keyword `Speech to Text` has three tokens. Keyword matching is case-insensitive, so `Speech to Text` is effectively equivalent to `speech to text`.

  For US English, the service normalizes each keyword to match spoken versus written strings. For example, it normalizes numbers to match how they are spoken as opposed to written. For other languages, keywords must be specified as they are spoken.

- Use the `keywords_threshold` parameter to specify a probability between 0.0 and 1.0 for a keyword match. The threshold indicates the lower bound for the level of confidence the service must have for a word to match the keyword. A keyword is spotted in the transcript only if its confidence is greater than or equal to the specified threshold.

  Specifying a small threshold can potentially produce many matches. If you specify a threshold, you must also specify one or more keywords. Omit the parameter to return no matches.

The following limits apply to keyword spotting:

- You can spot a maximum of 1000 keywords with a single request.
- A single keyword can have a maximum length of 1024 characters. The maximum effective length for double-byte languages might be shorter.
- Most HTTP servers and proxies impose a limit of 8 KB on the parameters of a request. Spotting a very large number of keywords or many lengthy keywords can exceed this limit. If you need to match more keywords, consider using a multipart HTTP request.

Keyword spotting is necessary to identify keywords in an audio stream. You cannot identify keywords by processing a final transcript because the transcript represents the service's best decoding results for the input audio. It does not include tokens with lower confidence scores that might represent a word of interest. So applying text processing tools to a transcript on the client side might not identify keywords. A richer representation of decoding results is needed, and that representation is available only at the server.

Keyword spotting results

The service returns the results in a `keywords_result` field that is an element of the `results` array. The `keywords_result` field is a dictionary, or associative array, of enumerable properties. Each property is identified by a specified keyword and includes an array of objects. The service returns one element of the array for each match that it finds for the keyword. The object for each match includes the following fields:

- `normalized_text` is the specified keyword that is normalized to the spoken phrase that matched in the input audio.
- `start_time` is the start time in seconds of the match.
- `end_time` is the end time in seconds of the match.
- `confidence` is the service's confidence that the match represents the specified keyword. The confidence must be at least as great as the specified threshold to be included in the results.

A keyword for which the service finds no matches is omitted from the array. A keyword might not be found if

- The audio simply does not include the keyword. Absence of the keyword is the most obvious explanation.

- The threshold is set too high. The service might identify the keyword but with a lower level of confidence, in which case it omits the match from the results.

- A keyword string that contains multiple tokens (for example, `Speech to Text`) is spoken with too much silence between its tokens. When the service transcribes audio, it chops the stream into a series of blocks. Each block represents a continuous chunk of audio that does not have an interval of silence that exceeds half of a second. It constructs an array of result objects that consists of these blocks.

  The service matches a multi-token keyword only if

  - The keyword's tokens are in the same block.
  - The tokens are either adjacent or separated by a gap of no more than 0.1 seconds.

  In the latter case, a brief filler or non-lexical utterance, such as "uhm" or "well," can lie between two tokens of the keyword. In this case, previous-generation models can insert a hesitation marker at this position of the transcript. Next-generation models include the actual hesitation in the transcription results. For more information, see Speech hesitations and hesitation markers.

Keyword spotting example

The following example request sets the `keywords` parameter to a URL-encoded array of three strings (`colorado`, `tornado`, and `tornadoes`) and the `keywords_threshold` parameter to `0.5`:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?keywords=colorado%2Ctornado%2Ctornadoes&keywords_threshold=0.5"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?keywords=colorado%2Ctornado%2Ctornadoes&keywords_threshold=0.5"
```

The service finds qualifying occurrences of `colorado` and `tornadoes`:

```
{
  "result_index": 0,
  "results": [
    {
      "keywords_result": {
        "colorado": [
          {
            "normalized_text": "Colorado",
            "start_time": 4.94,
            "confidence": 0.91,
            "end_time": 5.62
          }
        ],
        "tornadoes": [
          {
            "normalized_text": "tornadoes",
            "start_time": 1.52,
            "confidence": 1.0,
            "end_time": 2.15
          }
        ]
      },
      "alternatives": [
        {
          "confidence": 0.96,
          "transcript": "several tornadoes touch down as a line of
severe thunderstorms swept through Colorado on Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

Word alternatives

Note: The `word_alternatives_threshold` parameter is supported only with previous-generation models, not with next-generation models.

The word alternatives feature (also known as *Confusion Networks*) reports hypotheses for acoustically similar alternatives for words of the input audio. For instance, the word `Austin` might be the best hypothesis for a word from the audio. But the word `Boston` is another possible hypothesis in the same time interval. Hypotheses share a common start and end time but have different spellings and usually different confidence scores.

By default, the service does not report word alternatives. To indicate that you want to receive alternative hypotheses, you use the `word_alternatives_threshold` parameter to specify a probability between 0.0 and 1.0. The threshold indicates the lower bound for the level of confidence the service must have in a hypothesis to return it as a word alternative. A hypothesis is returned only if its confidence is greater than or equal to the specified threshold.

You can think of word alternatives as the timeline for a transcript that is chopped into smaller intervals, or bins. Each bin can have one or more hypotheses with different spellings and confidence. The `word_alternatives_threshold` parameter controls the density of the results that the service returns. Specifying a small threshold can potentially produce many hypotheses.

Word alternatives results

The service returns the results in a `word_alternatives` field that is an element of the `results` array. The `word_alternatives` field is an array of objects, each of which provides the following fields for an alternative word:

- `start_time` indicates the time in seconds at which the word for which hypotheses are returned starts in the input audio.
- `end_time` indicates the time in seconds at which the word for which hypotheses are returned ends in the input audio.
- `alternatives` is an array of hypothesis objects. Each object includes a `confidence` that indicates the service's confidence score for the hypothesis and a `word` that identifies the hypothesis. The confidence must be at least as great as the specified threshold to be included in the results. The service orders the alternatives by confidence.

Word alternatives example

The following example request specifies a rather low `word_alternatives_threshold` of `0.10`. The simple input audio includes words with common homonyms and different possible acoustic interpretations.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?word_alternatives_threshold=0.10"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?word_alternatives_threshold=0.10"
```

Because of the low threshold, the service returns multiple hypotheses and confidence scores for some of the words, along with the start and end times of all words. The final transcript correctly recognizes the input audio.

```
{
   "result_index": 0,
   "results": [
      {
         "final": true,
         "alternatives": [
            {
               "transcript": "yes I ate that tuna ",
               "confidence": 0.82
            }
         ],
         "word_alternatives": [
            {
               "start_time": 0.0,
               "end_time": 0.31,
               "alternatives": [
                  {
                     "word": "yes",
                     "confidence": 1.0
                  }
               ]
            },
            {
               "start_time": 0.31,
               "end_time": 0.46,
               "alternatives": [
                  {
                     "word": "I",
                     "confidence": 1.0
                  }
               ]
            },
            {
               "start_time": 0.46,
               "end_time": 0.63,
               "alternatives": [
                  {
                     "word": "ate",
                     "confidence": 0.89
                  },
                  {
                     "word": "eat",
                     "confidence": 0.11
                  }
               ]
            },
            {
               "start_time": 0.63,
               "end_time": 0.77,
               "alternatives": [
                  {
                     "word": "that",
                     "confidence": 0.72
                  },
                  {
                     "word": "the",
                     "confidence": 0.27
                  }
               ]
            },
            {
               "start_time": 0.77,
               "end_time": 1.2,
               "alternatives": [
                  {
                     "word": "tuna",
                     "confidence": 0.94
                  }
               ]
            }
         ]
      }
   ]
}
```

Response formatting and filtering

The IBM Watson® Speech to Text service provides three features that you can use to parse transcription results. You can format a final transcript to include more conventional representations of certain strings and to include punctuation. You can redact sensitive numeric information from a final transcript, and you can filter profanity from most transcription results. All of these features are beta functionality and are restricted to certain languages.

Smart formatting Version 2

The new version of smart formatting feature is available for US English, Brazilian Portuguese, French, German, Castilian Spanish, Spanish Latin American, and French Canadian. It is also available for the en-WW_Medical_Telephony model when US English audio is recognized.

The new version:

- provides more flexibility in adding new languages and patterns compared to older smart formatting.
- uses a more sophisticated machine learning technique (Weighted Finite State Transducers) to identify entities in texts compared to the older version, which was rule-based approach.
- provides more accurate entity classification and formatting and also adds capability to define hierarchies using weights when same text can be identified as two different entity type.

The `smart_formatting` feature directs the service to convert the following strings into more conventional representations:

- Dates and times
- Integers, decimals, ordinals
- Alphanumeric sequences (of length > 2)
- Phone numbers
- Currency values
- Measures (`/km²`, `kg`, `mph`, `m³`, etc)
- Emails, URLs and IP addresses
- Credit Card Numbers (formatted as groups of 4 digits)
- Punctuations (as spoken in dictations)

To use the new smart formatting feature for US English, Brazilian Portuguese, French and German; set the parameter smart_formatting=true and smart_formatting_version=2.

Entity Patterns and Examples

US English

- Different spoken forms of dates are accepted, including dates just as numbers or names of months and use of `the` and `of` (`the twenty fifth of july twenty twelve`). The dates are formatted as `m/d/yyyy`.
- Times are identified by keywords or suffix, for example, timezones (for example, `est`, `eastern`), `am`, `pm`, `hours`, `o'clock`, `minutes past hour`.
- Phone numbers must be either `911` or a number that contains 10 digits and/or starts with the number `[+]1`.
- Currency symbols are substituted for strings in appropriate contextsfor, for example, `dollar`, `cent`, `euro`, `yen`. `cent` is optional after `dollar`, for example, `twelve dollars twenty five` and `twelve dollars twenty five cents` formatted as `$12.25`.
- Internet email addresses with common format (for example, `[alphanumeric+symbols]+ at [alphanumeric dot]+ domainname`) are smart formatted.
- Web URLs, both short and long form are formatted. It includes protocol (`http/s`), subdomain (`www`), ports (`443,80`) and paths (`/help/abc`).
- Most large integers are formatted as numeric sequences. When large numbers (million, billions) are spoken with a single group integers, quantity word `million/billion` is not converted for readability, for example, `fifty nine million` -> `59 million` but when the number is more complex, it is formatted as numeric digits, for example, `fifty nine million and one` -> `59000001`.
- Numbers less than 10 are not converted to digit to avoid odd formatting, for example, `You are one of them` -> `You are 1 of them`. But in other context like expressing currency, they are converted appropriately, for example, `Give me one dollar` -> `Give me $1`.
- Most of the punctuation symbols are added for special keywords that occur in appropriate places. When you use smart formatting, the service substitutes spoken or dictated punctuation symbols for the keyword strings.
    - `comma` (`,`), `period` (`.`), `question mark` (`?`), `exclamation point` (`!`), `semicolon` (`;`), `hyphen` (`-`).

Smart formatting Examples

The following table shows examples of final transcripts both with and without smart formatting. The transcripts are based on US English audio.

Entity Type	Without smart formatting	With smart formatting
Dates	july twenty fifth two thousand twelve	7/25/2012
	the twenty fifth of july twenty twelve	7/25/2012
	january the thirty first two thousand	1/31/2000
	zero five zero five nineteen eighty three	6/20/2014
	second quarter of twenty twenty two	Q2 2022
Times	it is two eleven eastern	it is 02:11 est
	we begin at oh seven hundred hours	we begin at 07:00

	quarter past one	01:15
	three o'clock	03:00
Numbers	The quantity is one million one hundred and one	The quantity is 1000101
	One point five is between one and two	1.5 is between 1 and 2
	It would cost five point two million	It would cost 5.2 million
	Its one hundred twenty first trial	Its 121st trial
Phone numbers	nine one four five five six eight three three one	914-556-8331
	plus one nine two three one two three five six seven eight	+1 923-123-5678
Currency values	You owe me four united states dollars and sixty nine cents	You owe me $4.69
	seventy five dollars sixty three	$75.63
	Dollar rose to one hundred and nine point seven nine yen	Dollar rose to ¥109.79
Email, URL, IP	I saw the story on w w w dot yahoo dot com	I saw the story on www.yahoo.com
	a b three hyphen s d d dash three at g mail dot com	ab3-sdd-3@gmail.com
	h t t p colon slash slash w w w dot c o m d a i l y n e w s dot a b slash s m	http://www.comdailynews.ab/sm
	two two five dot double five dot o dot forty five	225.55.0.45
Measures	two hundred kilometers per hour	200 km/h
	two kilo watt hours	2 kWh
Sequences	H F H nine nine three dot seven B	HFH993.7B
	a ten eighty p display	a 1080p display

Table 2. Smart formatting example transcripts

Brazilian-Portuguese

- For dates, `do` and `de` in the transcript is used as separators for day, month and year. `primeiro` is considered as 1st of the month. The dates are formatted as `DD/MM/YYYY`.
- Times are identified by keywords and prefix, for example, `às` `ao`, `à`, `da tarde` (`p.m.`), `da madrugada` (`a.m.`), `meia noite`, `meio dia`. Prefixes `às` `ao`, `à` are optional.
- Landline numbers must have 10 digits (2 digits country code and 8 digits number), mobile numbers are 9 digits with first digit as `9` with optional country code. Area codes are optional. Numbers are formatted as `+NN (NN) NNNN-NNNN and +NN (NN) 9NNNN-NNNN`.
- Brazilian real currency symbol is `R$`. Other Currency symbols are substituted for strings in appropriate contexts, for example, `dollar`, `cent`, `euro`, `yen`. `centavos` is optional after `reais` for example, `setenta e cinco dólares e sessenta e três` and `setenta e cinco dólares e sessenta e três centavos` formatted as `R$75,63`
- Internet email addresses with common format (for example, `[alphanumeric+symbols]+ arroba [alphanumeric ponto]+ domainname`) are smart formatted.
- Web URLs, both short and long form, are formatted. It includes protocol (`http/s`), subdomain (`www`), ports (`443,80`) and paths (`/help/abc`).
- Most large integer are formatted as numeric sequences. When large numbers (milhões, bilhões, etc) are spoken with a single group integers, quantity word `milhões/bilhões` is not converted for readability for example, `doze milhões` -> `12 milhões` but when the number is more complex, it's formatted as numeric digits for example, `doze milhões e um` -> `12000001`.
- Numbers less than 10 are not formatted to digits to avoid odd conversions, for example, `vivo em uma casa` --> `vivo em 1 casa`.

- Most of the punctuation symbols are added for special keywords that occur in appropriate places. When you use smart formatting, the service substitutes spoken/dictated punctuation symbols for the keyword strings.
  - `vírgula` (`,`), `ponto` (`.`), `ponto de interrogação` (`?`), `ponto de exclamação` (`!`), `ponto e vírgula` (`;`), `hífen` (`-`).

Smart formatting Examples

The following table shows examples of final transcripts both with and without smart formatting. The transcripts are based on US English audio.

Entity Type	Without smart formatting	With smart formatting
Dates	trinta e um de dezembro de mil novecentos e oitenta e oito	31/12/1988
	um do um de mil novecentos e oitenta e sete	01/01/1987
Times	quinze pro meio dia	11:45
	meio dia e meia hora	12:30
	ao meio dia e meio	ao 12:30
	às dez pras duas da madrugada	às 1:50 a.m.
	às quinze para a meia noite	às 23:45
Numbers	cento e quarenta e sete mil quatrocentos e cinquenta e um	147451
	um vírgula vinte e seis	1,26
	décimo primeiro	11º
Phone numbers	quatro cinco um dois três quatro cinco seis sete oito	(45) 1234-5678
	onze nove nove oito meia cinco quinze zero dois	(11) 99865-1502
	nove vinte e sete vinte e oito trinta e sete trinta e oito	92728-3738
	mais cinco cinco onze nove meia nove zero meia zero um quatro meia	+55 (11) 96906-0146
Currency values	vinte e cinco centavos	R$ 0,25
	vinte e nove dólares e cinquenta centavos	$ 29,50
	vinte e cinco centavos	R$ 0,25
Email, URL, IP	a ponto b c arroba g mail ponto com	a.bc@gmail.com
	dáblio dáblio dáblio ponto a b c ponto es barra e f g	www.abc.es/efg
	w w w ponto nvidia ponto com	www.nvidia.com
	noventa e oito ponto setenta e seis ponto noventa e oito ponto dezesseis	98.76.98.16
Measures	duzentos e quarenta e cinco quilômetros por hora	245 kph
	duzentos e quarenta e cinco metros por segundo	245 m/s
Sequences	d dezesseis três nove c hífen f noventa e oito	d1639c-f98

	Modelo f t doze x	Modelo ft12x

Table 2. Smart formatting example transcripts

French

- In dates, the ordinal `premier` is considered 1st of the month. The dates are formatted as `DD/MM/YYYY`.
- Times are identified by keywords and prefix, for example, `heures`, `de l'après-midi` or `du soir`, `du matin`, `midi`. Times are formatted as 24H clock : `HH h MM`
- Telephone numbers must have 9 or 10 digits (5 pairs of two digits). In cases where only one digit of the first pairing is admitted, assumes that the 0 was skipped. Numbers are formatted as `NN NN NN NN NN`.
- When the `de` or `d'` preposition is used to express currency, the currency symbol is not used to format. This usually occurs with large round numbers, for example, `un milliard d'euro` formatted as `1 milliard d'euro`.
- Internet email addresses with common format (for example, `[alphanumeric+symbols]+ arobase [alphanumeric point]+ domainname`) are smart formatted. `@` can be represented by any of these: `arobase`, `chez`, `at`, `à`.
- Cardinals less than nine are not converted (in order to avoid `j'ai un pomme` -> `j'ai 1 pomme` and any other odd conversions.
- For ordinals, 'siècles' are rendered in roman numerals when given an ordinal adjective. e `dix-neuvième siècle` -> `XIXᵉ siècle`.
- Formatting of Fractions is supported. For example, `un onzième` -> `1/11`.
- Most of the punctuation symbols are added for special keywords that occur in appropriate places. When you use smart formatting, the service substitutes spoken/dictated punctuation symbols for the keyword strings.
  - `virgule` (`,`), `point` (`.`), `point d'interrogation` (`?`), `point d'exclamation` (`!`), `point-virgule` (`;`), `trait d'union` (`-`).

Smart formatting Examples

The following table shows examples of final transcripts both with and without smart formatting. The transcripts are based on US English audio.

Entity Type	Without smart formatting	With smart formatting
Dates	vingt-quatre juillet deux-mille-treize	24/7/2013
	dix-huit mai dix-neuf cent trente	18/5/1930
Times	huit heures du matin	8 h
	onze heures cinquante-sept	11 h 57
	deux heures de l'après-midi	14 h
Numbers	cent quarante-sept mille quatre cent cinquante et une	147451
	moins vingt-cinq-mille-trente-sept	25037
	vingt-troisièmes	23ᵉˢ
	quatre et deux quatrièmes	4 2/4
Phone numbers	double neuf douze trente-deux trente trente	99 12 32 30 30
	deux douze trente-deux trente trente	02 12 32 30 30
Currency values	deux dollars vingt	2,20 $
	cinq euro et soixante	5,60 €
	quatre virgule quatre-vingt milliards d'euros	4,80 milliards d'euros
Email, URL, IP	a b trois point s d d point trois arobase g mail point com	ab3.sdd.3@gmail.com

	w w w point web point c o point f r	www.web.co.fr
	double neuf dot trente-deux dot trente dot trente	99.32.30.30
Measures	quarante-deux-mille-deux-cent-cinquante-neuf par mètre carré	42 259 /m²
	deux cents kilomètres heure	200 km/h
Sequences	le document numéro zéro deux trente-six vingt-quatre	le document numéro 023624
	r t x dix-huit t i	rtx18ti

Table 2. Smart formatting example transcripts

French-Canadian

- In dates, the ordinal `premier` is considered 1st of the month. The dates are formatted as `DD/MM/YYYY`.
- Times are identified by keywords and prefix e.g. `heures`, `de l'après-midi` or `du soir`, `du matin`, `midi`. Times are formatted as 24H clock : `HH h MM`
- Phone numbers must be either `911` or a number that contains 10 digits and/or starts with the number `[+]1`.
- Internet email addresses with common format (e.g. `[alphanumeric+symbols]+ arobase [alphanumeric point]+ domainname`) are smart formatted. `@` can be represented by any of these: `arobase`, `chez`, `at`, `à`.
- Cardinals below nine are not converted if they occur in midst of other text(in order to avoid `j'ai un pomme` -> `j'ai 1 pomme` and any other odd conversions). They are still formatted if they occur in isolation with no other text.
- Formatting of Fractions is supported. e.g. `un onzième` -> `1/11`
- Most of the punctuation symbols are added for special keywords that occur in appropriate places. When you use smart formatting, the service substitutes spoken/dictated punctuation symbols for the keyword strings.
  - `virgule` (`,`), `point` (`.`), `point d'interrogation` (`?`), `point d'exclamation` (`!`), `point-virgule` (`;`), `trait d'union` (`-`), etc.

Smart formatting Examples

The following table shows examples of final transcripts both with and without smart formatting. The transcripts are based on US English audio.

Entity Type	Without smart formatting	With smart formatting
Dates	vingt-quatre juillet deux-mille-treize	24/7/2013
	dix-huit mai dix-neuf cent trente	18/5/1930
Times	huit heures du matin	8 h
	onze heures cinquante-sept	11 h 57
	deux heures de l'après-midi	14 h
Numbers	cent quarante-sept mille quatre cent cinquante et une	147451
	moins vingt-cinq-mille-trente-sept	25037
	vingt-troisièmes	23es
	quatre et deux quatrièmes	4 2/4
Phone numbers	plus un cinq un quatre cinq cinq cinq un deux trois quatre	+1 (514) 555-1234
	cinq un quatre quatre six neuf deux un zéro zéro	02 12 32 30 30
Currency values	deux dollars vingt	2,20 $

	vingt dollars cinq	20,05 $
	quatre virgule quatre-vingt milliards d'euros	4,80 milliards d'euros
Email, URL, IP	a b trois point s d d point trois arobase g mail point com	ab3.sdd.3@gmail.com
	w w w point web point c o point f r	www.web.co.fr
	double neuf dot trente-deux dot trente dot trente	(514) 469-210
Measures	quarante-deux-mille-deux-cent-cinquante-neuf par mètre carré	42 259 /m²
	deux cents kilomètres heure	200 km/h
Sequences	le document numéro zéro deux trente-six vingt-quatre	le document numéro 023624
	r t x dix-huit t i	rtx18ti

Table 2. Smart formatting example transcripts

Spanish

- In dates, the ordinal `primero` is considered 1st of the month. The dates are formatted as `DD/MM/YYYY`.
- Times on the hour or time without an article followed by a suffix (indicating 'a.m.' or 'p.m.'), it will be converted .e.g `las dos pe eme`. Times are formatted as 24H clock : `HH h MM` or as 12H clock with a.m./p.m.
- Telephone numbers must have 8, 9 or 10 digits. Numbers are formatted as `NNNN NNNN` or `NNN NNN NNN` or `NNN NNN NNNN`
- Internet email addresses with common format (e.g. `[alphanumeric+symbols]+ arroba [alphanumeric punto]+ domainname`) are smart formatted.
- Cardinals below nine are not converted if they occur in midst of other text(in order to avoid `un gato en el camino` -> `1 gato en el camino` and any other odd conversions). They are still formatted if they occur in isolation with no other text.
- Formatting of Fractions is supported. e.g. `un décimo` -> `1/10`
- Most of the punctuation symbols are added for special keywords that occur in appropriate places. When you use smart formatting, the service substitutes spoken/dictated punctuation symbols for the keyword strings.
    - `punto` (`.`), `interrogación` (`?`), `exclamación` (`!`), `punto y coma` (`;`), `guion medio` (`-`), etc.

Smart formatting Examples

The following table shows examples of final transcripts both with and without smart formatting. The transcripts are based on US English audio.

Entity Type	Without smart formatting	With smart formatting
Dates	treinta y uno de diciembre de mil novecientos noventa y dos	31/12/1992
	dieciséis de septiembre dos mil dieciocho	16/09/2018
Times	las dieciséis cincuenta	las 16:50
	las dos a eme	las 2:00 a.m.
Numbers	mil novecientos cincuenta y ocho	1958
	once mil novecientos cincuenta y ocho	11958
	décima primera	11ª
	un cuarentiunavo	1/41
Phone numbers	nueve uno cuatro cinco cinco seis ocho tres tres uno	914 556 8331

	uno dos tres cuatro cinco seis siete ocho	1234 5678
Currency values	dos euros noventa centavos	€2,90
	doce euros y cinco centavos	€12,05
	nueve punto cinco millones de pesos	$9.5 millones
Email, URL	a b c arroba g mail punto a b c	abc@gmail.abc
	doble uve doble uve doble uve punto nvidia punto com	www.nvidia.com
Measures	tres metros cúbicos	3 m³
	dos kilómetros por hora	2 kph
Sequences	cero dos tres seis dos cuatro	023624
	r t x cero dos tres w	rtx023w

Table 2. Smart formatting example transcripts

German

- Date formatting has support for both numeric and name for months (for example, `zweiter` is same as `februar`). The dates are formatted as `DD.MM.YYYY`.
- Times are identified by keywords, for example, `nach` `uhr`, `vor`, `minuten`. Time is formatted as 24-hr clock : `HH:MM:SS`.
- Telephone numbers should have 3-4 digit area code starting with `0` followed by 8-digit number. Country code (+49) is optional. Area code should not start with `0` if country code is used. Numbers are formatted as `+49 [N]NN NNNNNNNN or 0[N]NN NNNNNNNN` .
- Most Currency symbols are substituted for strings in appropriate contexts, for example, `dollar`, `cent`, `euro`, `yen`.
- Internet email addresses with common format (for example, `[alphanumeric+symbols]+ ät [alphanumeric punkt]+ domainname`) are formatted.
- Web URLs, both short and long form, are formatted. It includes protocol (`http/s`), subdomain (`www`), ports (`443,80`) and paths (`/help/abc`)
- Cardinals less than nine are not converted in order to avoid odd or ambiguous conversions.
- Ordinals and fractions formatting are supported.
- Most of the punctuation symbols are added for special keywords that occur in appropriate places. When you use smart formatting, the service substitutes spoken/dictated punctuation symbols for the keyword strings.
  - `komma` (`,`), `punkt` (`.`), `fragezeichen` (`?`), `ausrufezeichen` (`!`), `semikolon` (`;`), `bindestrich` (`-`).

Smart formatting Examples

The following table shows examples of final transcripts both with and without smart formatting. The transcripts are based on US English audio.

Entity Type	Without smart formatting	With smart formatting
Dates	vierundzwanzigster juli zwei tausend dreizehn	24.07.2013
	dreizehnter zweiter zwei tausend zwanzig	13.02.2020
Times	vierundzwanziguhrzweiundzwanzig	24:22 Uhr
	acht uhr sieben	08:07 Uhr
	ein uhr eine minute eine sekunde	01:01:01 Uhr
Numbers	minus fünf und zwanzig tausend sieben und dreißig	-25037

	acht hundert achtzehn komma drei null drei	818,303
	fünfundzwanzigtausendeinhundertelftem	25111.
	drei zwei ein hundertstel	3 2/100
Phone numbers	null vier eins eins eins zwei drei vier eins zwei drei vier	0411 12341234
	plus vier neun vier eins eins eins zwei drei vier eins zwei drei vier	+49 411 12341234
Currency values	zwei komma null null null eins dollar	2,0001 $
	zweiundzwanzig cent	0,22 €
Email, URL, IP	a b drei bindestrich s d d bindestrich drei ät g mail punkt com	ab3-sdd-3@gmail.com
	h t t p s doppelpunkt slash slash w w w punkt a b c punkt com slash a b	https://www.abc.com/ab
	drei fünf punkt eins drei fünf punkt zwei vier punkt zwei vier	35.135.24.24
Measures	zwei kilometer pro stunde	2 km/h
	vier hundert vierzig milliliter	440 ml
Sequences	c b vier drei bindestrich fünf drei fünf zwei vier zwei punkt vier drei fünf	cb43-535242.435
	teilenummer f t strich zwölf p	teilenummer ft-12p

Table 2. Smart formatting example transcripts

Smart formatting V2 examples

The following example requests smart formatting with a recognition request by setting the `smart_formatting` parameter to `true`. The following sections show the effects of smart formatting on the results of a request.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?model=en-US_Telephony&smart_formatting=true&smart_formatting_version=2"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?model=en-US_Telephony&smart_formatting=true&smart_formatting_version=2"
```

Smart formatting

The smart formatting feature is beta functionality that is available for US English, Japanese, and Spanish (all dialects). It also also available for the `en-WW_Medical_Telephony` model when US English audio is recognized.

The `smart_formatting` parameter directs the service to convert the following strings into more conventional representations:

- Dates
- Times
- Series of digits and numbers

- Phone numbers
- Currency values (for US English and Spanish)
- Internet email and web addresses (for US English and Spanish)

You set the `smart_formatting` parameter to `true` to enable smart formatting. By default, the service does not perform smart formatting. The service applies smart formatting just before it returns the final results to the client, when text normalization is complete. The conversion makes the transcript more readable and promotes better post-processing of transcription results by representing these artifacts as they would normally be written.

What results does smart formatting affect?

Smart formatting impacts some transcription results and not others:

- Smart formatting affects only words in the `transcript` field of final results, those results for which the `final` field is `true`. It does not affect interim results, for which `final` is `false`.
- Smart formatting does not affect words in other fields of the response. For example, smart formatting is not applied to response data in the `timestamps` or `alternatives` fields.
- Speech hesitations, such as "uhm" and "uh", can adversely impact the conversion of phrases and strings by smart formatting for some languages. Previous-generation models produce hesitation markers to replace such hesitations in a transcript. Smart formatting has the following effect on hesitation markers for previous-generation models:
  - *For US English,* smart formatting suppresses hesitation markers from the `transcript` field for final results.
  - *For Japanese,* hesitation markers continue to appear in final results.
  - *For both US English and Japanese,* hesitation markers continue to appear in interim results.
  - *For Spanish,* the service does not produce hesitation markers for any results.

  Next-generation models do not produce hesitation markers. They instead include the actual hesitations in transcription results. Smart formatting has no effect on hesitations that are included by next-generation models. For more information, see Speech hesitations and hesitation markers.

Language differences

Smart formatting is based on the presence of obvious keywords in the transcript. Because of differences among the supported languages, smart formatting works slightly differently for each language. The following sections describe the strings and content that trigger smart formatting changes for US English and Spanish and for Japanese.

US English and Spanish

- Times are identified by keywords such as `AM`, `PM`, or `EST`.
- Military times are converted if they are identified by the keyword `hours` (*US English*) or `horas` (*Spanish*).
- Phone numbers must be either `911` or a number that contains 10 or 11 digits and starts with the number `1`.
- Currency symbols are substituted for the following strings in appropriate contexts:
  - *For US English,* dollar, cent, and euro.
  - *For Spanish,* dolar, peso, peseta, libras esterlinas, libra, and euro.
- Internet email addresses are converted in some cases. Specifically, the service converts email addresses if the input audio uses the phrasing `email address ... {address}`. The following examples show a correct conversion of spoken phrases:
  - `My email address is j dot d o e at i b m dot com` becomes `My email address is j.doe@ibm.com`.
  - `Mi correo electronico es j punto d o e arroba i b m punto com` becomes `Mi correo electronico es j.doe@ibm.com`.
- Internet web addresses are converted in their short forms. Fully qualified web addresses are not converted. The following examples show complete conversions:
  - `I saw the story on yahoo dot com` becomes `I saw the story on yahoo.com`.
  - `Vi la historia en yahoo punto com` becomes `Vi la historia en yahoo.com`.

  The following examples show incomplete conversions:
  - `I saw the story on w w w dot yahoo dot com` becomes `I saw the story on w w w .yahoo.com`.
  - `Vi la historia en w w w punto yahoo punto com` becomes `Vi la historia en w w w .yahoo.com`.
- Conversion of large numbers and currency values can be challenging. The service converts digits and many numbers well. But larger and more complex numbers and currency values work best with more precise phrasing. For example, the service correctly converts the following transcripts

because of their precise wording:

  - `sixty nine thousand five hundred sixty dollars and twenty five cents` becomes `$69560.25`,
  - `sixty nine thousand five hundred sixty dollars point twenty five` becomes `$69560.25`.

  But the service cannot correctly convert the following transcripts due to their looser phrasing:

  - `sixty nine thousand five sixty dollars and twenty five cents` becomes `60 9000 $560.25`.
  - `sixty nine thousand five sixty dollars point twenty five` becomes `60 9000 $560.25`.

  To correctly convert a greater possible variety of complex numbers, you need to experiment with the results of smart formatting and customize your own post-processing utilities.

- *For US English,* certain punctuation symbols are added for special keywords that occur in appropriate places. When you use smart formatting, the service substitutes punctuation symbols for the following keyword strings based on where it finds them in a transcript:

  - `Comma` (`,`)
  - `Period` (`.`)
  - `Question mark` (`?`)
  - `Exclamation point` (`!`)

  The service converts these keyword strings to symbols only in appropriate positions of a transcript. In the following example, the speaker says the word `period` at the end of the sentence:

  - `the warranty period is short period` becomes `the warranty period is short.`

  The service correctly differentiates between the noun that appears earlier in the sentence and the concluding punctuation.

Japanese

- Phone numbers must be 10 or 11 digits and begin with valid prefixes for telephone numbers in Japan. For example, valid prefixes include `03` and `090`.
- English words are converted to ASCII (*hankaku*) characters. For example, `ＩＢＭ` is converted to `IBM`.
- Ambiguous terms might not be converted if sufficient context is unavailable. For example, it is unclear whether `一時` and `十分` refer to times.
- Punctuation is handled in the same way with or without smart formatting. For example, based on probability calculations, one of `カンマ` or `，` is selected.
- Strings that describe yen values are not replaced with the yen currency symbol.
- Internet email and web addresses in any form are not converted.
- The Japanese narrowband model (`ja-JP_NarrowbandModel`) includes some multigram word units for digits and decimal fractions. The service returns these multigram units regardless of whether you enable smart formatting. The following examples show the units that the service returns. The numbers in parentheses show the equivalent Arabic numeric expression for each unit.

  - *Digits:* `〇一` (01), ..., `〇九` (09), `一〇` (10), ..., `九〇` (90)
  - *Decimal fractions:* `〇・` (0.), `一・` (1.), ..., `十・` (10.)

  The smart formatting feature understands and returns the multigram units that the model generates. If you apply your own post-processing to transcription results, you need to handle these units appropriately.

Smart formatting results

The following table shows examples of final transcripts both with and without smart formatting. The transcripts are based on US English audio.

Information	Without smart formatting	With smart formatting
Dates	I was born on ten oh six nineteen seventy	I was born on 10/6/1970
	I was born on the ninth of December nineteen hundred	I was born on 12/9/1900
	Today is June sixth	Today is June 6
Times	The meeting starts at nine thirty AM	The meeting starts at 9:30 AM

	I am available at seven EST	I am available at 7:00 EST
	We meet at oh seven hundred hours	We meet at 0700 hours
Numbers	The quantity is one million one hundred and one	The quantity is 1000101
	One point five is between one and two	1.5 is between 1 and 2
Phone numbers	Call me at nine one four two three seven one thousand	Call me at 914-237-1000
	Call me at one nine one four nine oh nine twenty six forty five	Call me at 1-914-909-2645
Currency values	You owe me three thousand two hundred two dollars and sixty six	You owe me $3202.66
	The dollar rose to one hundred and nine point seven nine yen from one hundred and nine point seven two yen	The dollar rose to 109.79 yen from 109.72 yen
Internet email and web addresses	My email address is john dot doe at foo dot com	My email address is john.doe@foo.com
	I saw the story on yahoo dot com	I saw the story on yahoo.com
Combinations	The code is zero two four eight one and the date of service is May fifth two thousand and one	The code is 02481 and the date of service is 5/5/2001
	There are forty seven links on Yahoo dot com now	There are 47 links on Yahoo.com now

Table 2. Smart formatting example transcripts

Smart formatting results for long pauses

In cases where an utterance contains long enough pauses of silence, the service can split the transcript into two or more final results. This affects the contents of the response, as shown in the following examples.

Audio speech	Formatted transcription results
My phone number is nine one four five five seven three three nine two	"My phone number is 914-557-3392"
My phone number is nine one four ... *pause*... five five seven three three nine two	"My phone number is 914" "5573392"

Table 3. Smart formatting example transcripts for long pauses

For more information about specifying a pause interval that affects the service's response, see End of phrase silence time .

Smart formatting example

The following example requests smart formatting with a recognition request by setting the `smart_formatting` parameter to `true`. The following sections show the effects of smart formatting on the results of a request.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?smart_formatting=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?smart_formatting=true"
```

Numeric redaction

The numeric redaction feature is beta functionality that is available for US English, Japanese, and Korean.

The `redaction` parameter directs the service to redact, or mask, numeric data from final transcripts. The feature redacts any number that has three or more consecutive digits by replacing each digit with an `X` character. It is intended to redact sensitive numeric data, such as credit card numbers.

By default, the service does not redact numeric data. Set the `redaction` parameter to `true` to enable numeric redaction. When you enable redaction, the service automatically enables smart formatting by setting the `smart_formatting` parameter to `true`, regardless of whether you explicitly disable that feature. To ensure maximum security, the service also disables the following parameters when you enable redaction:

- The service disables keyword spotting, regardless of whether you specify values for the `keywords` and `keywords_threshold` parameters.
- The service disables maximum alternatives, regardless of whether you specify a value greater than 1 for the `max_alternatives` parameter. The service returns only a single final transcript.
- The service disables interim results for the WebSocket interface, regardless of whether you set the `interim_results` parameter to `true`.

The design of the feature parallels the existing smart formatting feature. The service applies redaction only to the final transcript of a recognition request, just before it returns the results to the client and after text normalization is complete.

Language differences

The feature works exactly as described for US English models but has the following differences for Japanese and Korean models.

Japanese

Japanese redaction has the following differences:

- In addition to masking strings of three or more consecutive digits, redaction also masks street addresses and numbers, regardless of whether they contain fewer than three digits.
- Similarly, redaction also masks date information in Japanese-style birth dates. In Japanese, date information is usually presented in Common Era format but sometimes follows Japanese style, particularly for birth dates. In this case, the year and month are masked even though they contain just one or two digits.

  For example, a Japanese-style birth date without redaction is `平成 30年 2月`. With redaction, the date becomes `平成 XX年 X月`.

Korean

Korean redaction has the following differences:

- The smart formatting feature is not supported. The service still performs numeric redaction for Korean, but it performs no other smart formatting.
- Isolated digit characters are reduced, but possible digit characters that are included as part of Korean phrases are not. For example, the character `이` in the following phrase is not replaced by an `X` because it is adjacent to the following character:

  `이입니다`

  If the `이` character were separated from the following character by a space, it would be replaced by an `X`, as described in Numeric redaction results.

Numeric redaction results

The following table shows examples of final transcripts both with and without numeric redaction in each supported language.

Language	Without redaction	With redaction
US English	my credit card number is four one four seven two	my credit card number is XXXXX
Japanese	私 のクレジット カード 番号 は 四 一 四 七 二です	私 のクレジット カード 番号 は XXXXX です
Korean	내 신용 카드 번호는 사 일 사 칠 이 번입니다	내 신용 카드 번호는 XXXXX 번입니다

Table 5. Numeric redaction example transcripts

Numeric redaction example

The following example requests numeric redaction with a recognition request by setting the `redaction` parameter to `true`. Because the request enables redaction, the service implicitly enables smart formatting with the request. The service effectively disables the other parameters of the request so that they have no effect: The service returns a single final transcript and recognizes no keywords.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?&redaction=true&max_alternatives=3&keywords=birth%2Cbirthday&keywords_threshold=0.5"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @{path}audio-file.wav \
"{url}/v1/recognize?&redaction=true&max_alternatives=3&keywords=birth%2Cbirthday&keywords_threshold=0.5"
```

Profanity filtering

Note: The profanity filtering feature is generally available for US English and Japanese only.

The `profanity_filter` parameter indicates whether the service is to censor profanity from its results. By default, the service obscures all profanity by replacing it with a series of asterisks in the transcript. Setting the parameter to `false` displays words in the output exactly as transcribed.

The service censors profanity from all final transcripts and from any alternative transcripts. It also censors profanity from results that are associated with word alternatives, word confidence, and word timestamps. The sole exception is keyword spotting, for which the service returns all words as specified by the user, regardless of whether `profanity_filter` is `true`.

Profanity filtering example

The following example shows the results for a brief audio file that is transcribed with the default `true` value for the `profanity_filter` parameter. The request also sets the `word_alternatives_threshold` parameter to a relatively high value of `0.99` and the `word_confidence` and `timestamps` parameters to `true`.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?word_alternatives_threshold=0.99&word_confidence=true&timestamps=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?word_alternatives_threshold=0.99&word_confidence=true&timestamps=true"
```

The service masks profanity from the response by replacing it with a series of asterisks:

```
{
  "result_index": 0,
  "results": [
    {
      "word_alternatives": [
        {
          "start_time": 0.03,
          "alternatives": [
            {
              "confidence": 1.0,
              "word": "****"
            }
          ],
          "end_time": 0.25
        },
        {
          "start_time": 0.25,
          "alternatives": [
            {
              "confidence": 0.99,
              "word": "you"
            }
          ],
          "end_time": 0.56
        }
      ],
      "alternatives": [
        {
          "transcript": "**** you",
          "confidence": 0.99,
          "word_confidence": [
            ["****", 1.0],
            ["you", 0.99]
          ],
          "timestamps": [
            ["****", 0.03, 0.25],
            ["you", 0.25, 0.56]
          ]
        }
      ],
      "final": true
    }
  ]
}
```

Response metadata

The IBM Watson® Speech to Text service can return three types of metadata about its transcription results. You can request maximum alternatives to see multiple possible final transcription results. You can also request word confident and word timestamps to receive confidence measures and timestamps for each word of the audio.

Maximum alternatives

The `max_alternatives` parameter accepts an integer value that tells the service to return the *n*-best alternative hypotheses for the results. By default, the service returns only a single transcription result, which is equivalent to setting the parameter to `1`. By setting `max_alternatives` to a number greater than 1, you ask the service to return that number of the best alternative transcriptions. (If you specify a value of `0`, the service uses the default value of `1`.)

The service reports a confidence score only for the best alternative that it returns. In most cases, that is the alternative to choose.

Note: Internal changes and improvements to the service can affect transcripts and confidence scores. For example, speech recognition may be improved to return more precise transcription results. Similarly, transcript and word confidence scores might change slightly as a result of improved speech recognition. Such changes are expected to be modest, but do not expect transcripts and confidence scores to remain unchanged over time.

Maximum alternatives example

The following example request sets the `max_alternatives` parameter to `3`:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?max_alternatives=3"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?max_alternatives=3"
```

The service reports a confidence only for the most likely of the three alternatives:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.96,
          "transcript": "several tornadoes touch down as a line of
severe thunderstorms swept through Colorado on Sunday "
        },
        {
          "transcript": "several tornadoes touched down as a line of
severe thunderstorms swept through Colorado on Sunday "
        },
        {
          "transcript": "several tornadoes touch down as a line of
severe thunderstorms swept through Colorado and Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

Word confidence

The `word_confidence` parameter tells the service whether to provide confidence measures for the words of the transcript. By default, the service reports a confidence measure only for the final transcript as a whole. Setting `word_confidence` to `true` directs the service to report a confidence measure for each individual word of the transcript.

A confidence measure indicates the service's estimation that the transcribed word is correct based on the acoustic evidence. Confidence scores range from 0.0 to 1.0.

- A score of 1.0 indicates that the current transcription of the word reflects the most likely result.
- A score of 0.5 means that the word has a 50-percent chance of being correct.

Note: Internal changes and improvements to the service can affect transcripts and confidence scores. For example, speech recognition may be improved to return more precise transcription results. Similarly, transcript and word confidence scores might change slightly as a result of improved speech recognition. Such changes are expected to be modest, but do not expect transcripts and confidence scores to remain unchanged over time.

Word confidence example

The following example requests word confidence scores for the words of the transcription:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?word_confidence=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?word_confidence=true"
```

The service returns a confidence score for each word of the audio:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.96,
          "transcript": "several tornadoes touch down as a line of
severe thunderstorms swept through Colorado on Sunday ",
          "word_confidence": [
            [
              "several",
              1.0
            ],
            [
              "tornadoes",
              1.0
            ],
            [
              "touch",
              0.52
            ],
            [
              "down",
              0.90
            ],
            . . .
            [
              "on",
              0.31
            ],
            [
              "Sunday",
              0.99
            ]
          ]
        }
      ],
      "final": true
    }
  ]
}
```

Word timestamps

The `timestamps` parameter tells the service whether to produce timestamps for the words it transcribes. By default, the service reports no timestamps. Setting `timestamps` to `true` directs the service to report the beginning and ending time in seconds for each word relative to the start of the audio.

Timestamps are automatically enabled when you request speaker labels. For more information, see Speaker labels.

Word timestamps example

The following example requests timestamps for the words of the transcription:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?timestamps=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?timestamps=true"
```

The service returns a timestamp for each word of the audio:

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "timestamps": [
            [
              "several",
              1.01,
              1.52
            ],
            [
              "tornadoes",
              1.52,
              2.15
            ],
            [
              "touch",
              2.15,
              2.5
            ],
            [
              "down",
              2.5,
              2.81
            ],
            . . .
            [
              "on",
              5.62,
              5.74
            ],
            [
              "Sunday",
              5.74,
              6.34
            ]
          ],
          "confidence": 0.96,
          "transcript": "several tornadoes touch down as a line of
severe thunderstorms swept through Colorado on Sunday "
        }
      ],
      "final": true
    }
  ]
}
```

Processing and audio metrics

The IBM Watson® Speech to Text service can return two types of optional metrics for a speech recognition request:

- Processing metrics provide periodic information about the service's processing of the input audio. Use the metrics to gauge the service's progress in transcribing the audio.
- Audio metrics provide information about the signal characteristics of the input audio. Use the metrics to determine the characteristics and quality of the audio.

By default, the service returns no processing or audio metrics for a request.

Processing metrics

Note: The `processing_metrics` and `processing_metrics_interval` parameters are supported only with previous-generation models, not with next-generation models. Also, processing metrics are available only with the WebSocket and asynchronous HTTP interfaces. They are not available with the synchronous HTTP interface.

Processing metrics provide detailed timing information about the service's analysis of the input audio. The service returns the metrics at specified intervals and with transcription events, such as interim and final results.

The metrics include statistics about how much audio the service has received, how much audio the service has transferred to the speech recognition engine, and how long the service has been processing the audio. If you request speaker labels, the information also shows how much audio the service has processed to determine speaker labels.

Processing metrics can help you gauge the progress of a recognition request. They can also help you distinguish the absence of results due to

- Lack of audio.
- Lack of speech in submitted audio.
- Engine stalls at the server and network stalls between the client and the server. To differentiate between engine and network stalls, results are periodic rather than event-based.

The metrics can also help you estimate response jitter by examining the periodic arrival times. Metrics are generated at a constant interval, so any difference in arrival times is caused by jitter.

Requesting processing metrics

To request processing metrics, use the following optional parameters:

- `processing_metrics` is a boolean that indicates whether the service is to return processing metrics. Specify `true` to request the metrics. By default, the service returns no metrics.

- `processing_metrics_interval` is a float that specifies the interval in seconds of real wall-clock time at which the service is to return metrics. By default, the service returns metrics once per second. The service ignores this parameter unless the `processing_metrics` parameter is set to `true`.

  The parameter accepts a minimum value of 0.1 seconds. The level of precision is not restricted, so you can specify values such as 0.25 and 0.125. The service does not impose a maximum value.

How you provide the parameters and how the service returns processing metrics differ by interface:

- With the WebSocket interface, you specify the parameters with the JSON `start` message for a speech recognition request. The service calculates and returns metrics in real-time at the requested interval.
- With the asynchronous HTTP interface, you specify query parameters with a speech recognition request. The service calculates the metrics at the requested interval, but it returns all metrics for the audio with the final transcription results.

The service also returns processing metrics for transcription events. If you request interim results with the WebSocket interface, you can receive metrics with greater frequency than the requested interval.

To receive processing metrics only for transcription events instead of at periodic intervals, set the processing interval to a large number. If the interval is larger than the duration of the audio, the service returns processing metrics only for transcription events.

Understanding processing metrics

The service returns processing metrics for transcription events in the `processing_metrics` field of the `SpeechRecognitionResults` object. For metrics generated at periodic intervals, not with transcription events, the object contains only the `processing_metrics` field, as shown in the following example.

```
{
  "processing_metrics": {
    "processed_audio": {
      "received": float,
      "seen_by_engine": float,
      "transcription": float,
      "speaker_labels": float
    },
    "wall_clock_since_first_byte_received": float,
    "periodic": boolean
  }
}
```

The `processing_metrics` field includes a `ProcessingMetrics` object that has the following fields:

- `wall_clock_since_first_byte_received` is the amount of real time in seconds that has passed since the service received the first byte of input audio. Values in this field are generally multiples of the specified metrics interval, with two differences:
    - Values might not reflect exact intervals such as 0.25, 0.5, and so on. Actual values might, for example, be 0.27, 0.52, and so on, depending on when the service receives and processes audio.
    - Values for transcription events are not related to the processing interval. The service returns event-driven results as they occur.
- `periodic` indicates whether the metrics apply to a periodic interval or to a transcription event:
    - `true` means that the response was triggered by a processing interval. The information contains processing metrics only.
    - `false` means that the response was triggered by a transcription event. The information contains processing metrics plus transcription results.

  Use this field to identify why the service generated the response and to filter different results if necessary.
- `processed_audio` includes a `ProcessedAudio` object that provides detailed timing information about the service's processing of the input audio.

The `ProcessedAudio` object includes the following fields. All of the fields refer to seconds of audio as of this response. Only the `wall_clock_since_first_byte_received` field refers to elapsed real-time.

- `received` is the seconds of audio that the service has received.
- `seen_by_engine` is the seconds of audio that the service has passed to its speech processing engine.
- `transcription` is the seconds of audio that the service has processed for speech recognition.
- `speaker_labels` is the seconds of audio that the service has processed for speaker labels. The response includes this field only if you request speaker labels.

The speech processing engine analyzes the input audio multiple times. The `processed_audio` object shows values for audio that the engine has processed and will not read again. Processed audio has no effect on future recognition hypotheses.

The metrics indicate the progress and complexity of the engine's processing:

- `seen_by_engine` is audio that the service has read and passed to the engine at least once.
- `received` - `seen_by_engine` is audio that has been buffered at the service but has not yet been seen or processed by the engine.
- The relationship between the times is `received` >= `seen_by_engine` >= `transcription` >= `speaker_labels`.

The following relationships can also be helpful in understanding the results:

- The values of the `received` and `seen_by_engine` fields are greater than the values of the `transcription` and `speaker_labels` fields during speech recognition processing. The service must receive the audio before it can begin to process results.
- The values of the `received` and `seen_by_engine` fields are identical when the service has finished processing the audio. The final values of the fields can be greater than the values of the `transcription` and `speaker_labels` fields by a fractional number of seconds.
- The value of the `speaker_labels` field typically trails the value of the `transcription` field during speech recognition processing. The values of the `transcription` and `speaker_labels` fields are identical when the service has finished processing the audio.

Processing metrics example: WebSocket interface

The following example shows the `start` message that is passed for a request to the WebSocket interface. The request enables processing metrics and sets the processing metrics interval to 0.25 seconds. It also sets the `interim_results` and `speaker_labels` parameters to `true`. The audio contains the simple message "hello world long pause stop."

```
function onOpen(evt) {
  var message = {
    action: 'start',
    content-type: 'audio/flac',
    processing_metrics: true,
    processing_metrics_interval: 0.25,
    interim_results: true,
    speaker_labels: true
  };
  websocket.send(JSON.stringify(message));
  websocket.send(blob);
}
```

The following example output shows the first few processing metrics results that the service returns for the request.

```
{
  "processing_metrics": {
    "processed_audio": {
      "received": 7.04,
      "seen_by_engine": 1.59,
      "transcription": 0.49,
      "speaker_labels": 0.0
    },
    "wall_clock_since_first_byte_received": 0.32,
    "periodic": true
  }
}
{
  "processing_metrics": {
    "processed_audio": {
      "received": 7.04,
      "seen_by_engine": 1.59,
      "transcription": 0.49,
      "speaker_labels": 0.0
    },
    "wall_clock_since_first_byte_received": 0.51,
    "periodic": false
  },
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "timestamps": [
            [
              "hello",
              0.43,
              0.76
            ],
            [
              "world",
              0.76,
              1.22
            ]
          ],
          "transcript": "hello world "
        }
      ],
      "final": false
    }
  ]
}
{
  "processing_metrics": {
    "processed_audio": {
      "received": 7.04,
      "seen_by_engine": 2.59,
      "transcription": 1.25,
      "speaker_labels": 0.0
    },
    "wall_clock_since_first_byte_received": 0.57,
    "periodic": true
  }
}
. . .
```

Processing metrics example: Asynchronous HTTP interface

The following example shows a speech recognition request for the `/v1/recognitions` method of the asynchronous HTTP interface. The request enables processing metrics and specifies an interval of 0.25 seconds. The audio file again includes the message "hello world long pause stop".

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognitions?processing_metrics=true&processing_metrics_interval=0.25"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognitions?processing_metrics=true&processing_metrics_interval=0.25"
```

The following example output shows the first two processing metrics results that the service returns for the request.

```
{
  "processing_metrics": {
    "processed_audio": {
      "received": 7.04,
      "seen_by_engine": 1.59,
      "transcription": 0.49
    },
    "wall_clock_since_first_byte_received": 0.32,
    "periodic": true
  }
}
{
  "processing_metrics": {
    "processed_audio": {
      "received": 7.04,
      "seen_by_engine": 2.59,
      "transcription": 1.25
    },
    "wall_clock_since_first_byte_received": 0.57,
    "periodic": true
  }
}
. . .
```

Audio metrics

Audio metrics provide detailed information about the signal characteristics of the input audio. The results provide aggregated metrics for the entire input audio at the conclusion of speech processing. For a technically sophisticated user, the metrics can provide meaningful insight into the detailed characteristics of the audio.

You can use audio metrics to provide a real-time indication of a problem with the input audio and possibly even a potential solution. For example, you can provide a message such as "There is too much noise in the background" or "Please come closer to the microphone." You can also use an offline analytical tool to review the signal characteristics and suggest data that is suitable for future model updates.

Requesting audio metrics

To request audio metrics, set the `audio_metrics` boolean parameter to `true`. By default, the service returns no metrics.

- With the WebSocket interface, you specify the parameter with the JSON `start` message for a speech recognition request.
- With the HTTP interfaces, you specify a query parameter with a speech recognition request.

The service returns audio metrics with the final transcription results. It returns the metrics just once for the entire audio stream. Even if the service returns multiple transcription results (for different blocks of audio), it returns only a single instance of the metrics at the very end of the results.

The WebSocket interface accepts multiple audio streams (or files) with a single connection. You delimit different streams by sending `stop` messages or empty binary blobs to the service. In this case, the service returns separate metrics for each delimited audio stream.

Understanding audio metrics

The service return the metrics in the `audio_metrics` field of the `SpeechRecognitionResults` object.

```
{
  "results": [
    . . .
  ],
  "result_index": integer,
  "audio_metrics": {
    "sampling_interval": float,
    "accumulated": {
      "final": boolean,
      "end_time": float,
      "signal_to_noise_ratio": float,
      "speech_ratio": float,
      "high_frequency_loss": float,
      "direct_current_offset": [
        {"begin": float, "end": float, "count": integer},
        . . .
      ],
      "clipping_rate": [
        {"begin": float, "end": float, "count": integer},
        . . .
      ],
      "speech_level": [
        {"begin": float, "end": float, "count": integer},
        . . .
      ],
      "non_speech_level": [
        {"begin": float, "end": float, "count": integer},
        . . .
      ]
    }
  }
}
```

The `audio_metrics` field includes an `AudioMetrics` object that has two fields:

- `sampling_interval` indicates the interval in seconds (typically 0.1 seconds) at which the service calculated the audio metrics.
- `accumulated` includes an `AudioMetricsDetails` object that provides detailed information about the signal characteristics of the input audio.

The following fields of the `AudioMetricsDetails` object provide unary values:

- `final` indicates whether the metrics are for the end of the audio stream, meaning that transcription is complete. Currently, the field is always `true`. The service returns metrics just once per audio stream. The results provide aggregated metrics for the complete stream.
- `end_time` specifies the end time in seconds of the audio to which the metrics apply. Because the metrics apply to the entire audio stream, the end time is always the length of the audio.
- `signal_to_noise_ratio` provides the signal-to-noise ratio (SNR) for the audio signal. The value indicates the ratio of speech to noise in the audio. A valid value lies in the range of 0 to 100 decibels (dB). The service omits the field if it cannot compute the SNR for the audio.
- `speech_ratio` is the ratio of speech to non-speech segments in the audio signal. The value lies in the range of 0.0 to 1.0.
- `high_frequency_loss` indicates the probability that the audio signal is missing the upper half of its frequency content.
  - A value close to 1.0 typically indicates artificially up-sampled audio, which negatively impacts the accuracy of the transcription results.
  - A value at or near 0.0 indicates that the audio signal is good and has a full spectrum.
  - A value around 0.5 means that detection of the frequency content is unreliable or unavailable.

The following fields of the `AudioMetricsDetails` object provide histograms for signal characteristics. Each field provides an array of `AudioMetricsHistogramBin` objects. A single unit in each histogram is calculated based on a `sampling_interval` length of audio.

- `direct_current_offset` defines a histogram of the cumulative direct current (DC) component of the audio signal.
- `clipping_rate` defines a histogram of the clipping rate for the audio segments. The clipping rate is defined as the fraction of samples in the segment that reach the maximum or minimum value that is offered by the audio quantization range.

  The service auto-detects either a 16-bit Pulse-Code Modulation (PCM) audio range (-32768 to +32767) or a unit range (-1.0 to +1.0). The clipping rate is between 0.0 and 1.0. Higher values indicate possible degradation of speech recognition.
- `speech_level` defines a histogram of the signal level in segments of the audio that contain speech. The signal level is computed as the Root-Mean-Square (RMS) value in a decibel (dB) scale normalized to the range 0.0 (minimum level) to 1.0 (maximum level).
- `non_speech_level` defines a histogram of the signal level in segments of the audio that do not contain speech. The signal level is computed as the RMS value in a decibel scale normalized to the range 0.0 (minimum level) to 1.0 (maximum level).

Each `AudioMetricsHistogramBin` object describes a bin with defined `begin` and `end` boundaries. Each bin indicates the `count` or number of values in the range of signal characteristics for that bin. The first and last bins of a histogram are the boundary bins. They cover the intervals between negative infinity and the first boundary, and between the last boundary and positive infinity, respectively.

Audio metrics example

The following example shows a speech recognition request with the synchronous HTTP interface that returns audio metrics. The audio file includes the simple message "hello world long pause stop".

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?audio_metrics=true"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?audio_metrics=true"
```

The response includes the audio metrics for the complete input audio, which has a duration of 7.0 seconds. The input audio has slightly more speech than non-speech segments: 37 to 33 segments, for a `speech_ratio` of 0.529. The clipping rate is very low, indicating high-quality input audio.

The `high_frequency_loss` field has a value of 0.5, meaning that the service's detection of the frequency content is unreliable or unavailable. The results omit the `signal_to_noise_ratio` field because the service could not calculate the SNR for the audio.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.96,
          "transcript": "hello world "
        }
      ],
      "final": true
    },
    {
      "alternatives": [
        {
          "confidence": 0.79,
          "transcript": "long pause "
        }
      ],
      "final": true
    },
    {
      "alternatives": [
        {
          "confidence": 0.97,
          "transcript": "stop "
        }
      ],
      "final": true
    }
  ],
  "audio_metrics": {
    "sampling_interval": 0.1,
    "accumulated": {
      "final": true,
      "end_time": 7.0,
      "speech_ratio": 0.529,
      "high_frequency_loss": 0.5,
      "direct_current_offset": [
        {"begin": -1.0, "end": -0.9, "count": 0},
        {"begin": -0.9, "end": -0.7, "count": 0},
        {"begin": -0.7, "end": -0.5, "count": 0},
        {"begin": -0.5, "end": -0.3, "count": 0},
        {"begin": -0.3, "end": -0.1, "count": 0},
        {"begin": -0.1, "end": 0.1, "count": 70},
        {"begin": 0.1, "end": 0.3, "count": 0},
        {"begin": 0.3, "end": 0.5, "count": 0},
        {"begin": 0.5, "end": 0.7, "count": 0},
        {"begin": 0.7, "end": 0.9, "count": 0},
        {"begin": 0.9, "end": 1.0, "count": 0}
      ],
      "clipping_rate": [
        {"begin": 0.0, "end": 1e-05, "count": 70},
        {"begin": 1e-05, "end": 0.0001, "count": 0},
        {"begin": 0.0001, "end": 0.001, "count": 0},
        {"begin": 0.001, "end": 0.01, "count": 0},
        {"begin": 0.01, "end": 0.1, "count": 0},
        {"begin": 0.1, "end": 1.0, "count": 0}
      ],
      "speech_level": [
        {"begin": 0.0, "end": 0.1, "count": 37},
        {"begin": 0.1, "end": 0.2, "count": 0},
        {"begin": 0.2, "end": 0.3, "count": 0},
        {"begin": 0.3, "end": 0.4, "count": 0},
        {"begin": 0.4, "end": 0.5, "count": 0},
        {"begin": 0.5, "end": 0.6, "count": 0},
        {"begin": 0.6, "end": 0.7, "count": 0},
        {"begin": 0.7, "end": 0.8, "count": 0},
        {"begin": 0.8, "end": 0.9, "count": 0},
        {"begin": 0.9, "end": 1.0, "count": 0}
      ],
      "non_speech_level": [
        {"begin": 0.0, "end": 0.1, "count": 33},
        {"begin": 0.1, "end": 0.2, "count": 0},
        {"begin": 0.2, "end": 0.3, "count": 0},
        {"begin": 0.3, "end": 0.4, "count": 0},
        {"begin": 0.4, "end": 0.5, "count": 0},
        {"begin": 0.5, "end": 0.6, "count": 0},
        {"begin": 0.6, "end": 0.7, "count": 0},
        {"begin": 0.7, "end": 0.8, "count": 0},
        {"begin": 0.8, "end": 0.9, "count": 0},
        {"begin": 0.9, "end": 1.0, "count": 0}
      ]
    }
  }
}
```

Parameter summary

The following sections provide a summary of all of the parameters that are available for speech recognition. The information includes availability for large speech models, previous- and next-generation models, and support and usage for speech recognition interfaces.

- For more information about previous-generation languages and models, see Previous-generation languages and models.
- For more information about next-generation languages and models, see Next-generation languages and models.
- For more information about large speech languages and models, see Large speech languages and models;

access_token

A required access token that you use to establish an authenticated connection with the WebSocket interface. For more information, see Open a connection.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Query parameter of `/v1/recognize` connection request

Synchronous HTTP	Not supported
Asynchronous HTTP	Not supported

Table 1. The access_token parameter

acoustic_customization_id

An optional customization ID for a custom acoustic model that is adapted for the acoustic characteristics of your environment and speakers. By default, no custom model is used. For more information, see Using a custom acoustic model for speech recognition .

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available or beta for all models that support acoustic model customization. For more information, see Customization support for previous-generation models.
Next-generation models	Not available.
WebSocket	Query parameter of `/v1/recognize` connection request
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 2. The acoustic_customization_id parameter

audio_metrics

An optional boolean that indicates whether the service returns metrics about the signal characteristics of the input audio. By default (`false`), the service returns no audio metrics. For more information, see Audio metrics.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 3. The audio_metrics parameter

background_audio_suppression

An optional float between 0.0 and 1.0 that indicates the level to which background audio and side conversations are to be suppressed in the input audio. The default is 0.0, which provides no suppression of background audio. For more information, see Background audio suppression.

Availability and usage	Description

Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all language models except for `ar-MS_BroadbandModel`, `pt-BR_BroadbandModel`, `zh-CN_BroadbandModel`, `zh-CN_NarrowbandModel`, and `de-DE_BroadbandModel`.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 4. The background_audio_suppression parameter

base_model_version

An optional version of a base model. The parameter is intended primarily for use with custom models that are updated for a new base model, but it can be used without a custom model. The default value depends on whether the parameter is used with or without a custom model. For more information, see Making speech recognition requests with upgraded custom models .

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Query parameter of `/v1/recognize` connection request
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 5. The base_model_version parameter

character_insertion_bias

An optional float between -1.0 and 1.0 that indicates whether the service is biased to recognize shorter (negative values) or longer (positive values) strings of characters when developing transcription hypotheses. By default, the service uses a default bias of 0.0. The value that you specify represents a change from a model's default. For more information, see Character insertion bias.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Not available.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method

Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 6. The character_insertion_bias parameter

Content-Type

An optional audio format (MIME type) that specifies the format of the audio data that you pass to the service. The service can automatically detect the format of most audio, so the parameter is optional for most formats. It is required for the `audio/alaw`, `audio/basic`, `audio/l16`, and `audio/mulaw` formats. For more information, see Specifying an audio format.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	`content-type` parameter of JSON `start` message
Synchronous HTTP	Request header of `POST /v1/recognize` method
Asynchronous HTTP	Request header of `POST /v1/recognitions` method

Table 7. The Content-Type parameter

customization_weight

An optional double between 0.0 and 1.0 that indicates the relative weight that the service gives to words from a custom language model versus words from the base vocabulary. The default differs for different types of models. You can specify a value either when a custom model is trained or when it is used for speech recognition. For more information, see Using customization weight.

Availability and usage	Description
Large speech models	Generally available for all languages. The default value is 0.5.
Previous-generation models	Generally available or beta for all models that support language model customization. The default value is 0.3. For more information, see Customization support for previous-generation models.
Next-generation models	Generally available or beta for all models that support language model customization. The default value is 0.2 for most next-generation models; it is 0.1 for models that are based on new language model customization technology. For more information, see Customization support for next-generation models.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 8. The customization_weight parameter

end_of_phrase_silence_time

An optional double between 0.0 and 120.0 that indicates the pause interval at which the service splits a transcript into multiple final results if it encounters silence. By default, the service uses a pause interval of 0.8 seconds for all languages other than Chinese, for which it uses an interval of 0.6 seconds. For more information, see End of phrase silence time.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 9. The end_of_phrase_silence_time parameter

grammar_name

An optional string that identifies a grammar that is to be used for speech recognition. The service recognizes only strings that are defined by the grammar. You must specify both the name of the grammar and the customization ID of the custom language model for which the grammar is defined. For more information, see Using a grammar for speech recognition .

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available or beta for all models that support language model customization. For more information, see Customization support for previous-generation models.
Next-generation models	Generally available or beta for all models that support language model customization. For more information, see Customization support for next-generation models.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 10. The grammar_name parameter

inactivity_timeout

An optional integer that specifies the number of seconds for the service's inactivity timeout. Inactivity means that the service detects no speech in streaming audio. The default is 30 seconds. Use `-1` to indicate infinity. For more information, see Inactivity timeout.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method

Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 11. The inactivity_timeout parameter

interim_results

An optional boolean that directs the service to return intermediate hypotheses that are likely to change before the final transcript. By default (`false`), interim results are not returned. Interim results are available only with the WebSocket interface. For more information, see Interim results.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for next-generation models that support low latency, but only if both the `interim_results` and `low_latency` parameters are set to `true`. For more information, see Requesting interim results and low latency.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Not supported
Asynchronous HTTP	Not supported

Table 12. The interim_results parameter

keywords

An optional array of keyword strings that the service spots in the input audio. By default, keyword spotting is not performed. For more information, see Keyword spotting.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available for all languages.
Next-generation models	Not available.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 13. The keywords parameter

keywords_threshold

An optional double between 0.0 and 1.0 that indicates the minimum threshold for a positive keyword match. By default, keyword spotting is not performed. For more information, see Keyword spotting.

Availability and usage	Description

Large speech models	Not available.
Previous-generation models	Generally available for all languages.
Next-generation models	Not available.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 14. The keywords_threshold parameter

language_customization_id

An optional customization ID for a custom language model that includes terminology from your domain. By default, no custom model is used. For more information, see Using a custom language model for speech recognition .

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available or beta for all models that support language model customization. For more information, see Customization support for previous-generation models.
Next-generation models	Generally available or beta for all models that support language model customization. For more information, see Customization support for next-generation models.
WebSocket	Query parameter of `/v1/recognize` connection request
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 15. The language_customization_id parameter

low_latency

An optional boolean that indicates whether the service is to produce results more quickly at the possible expense of transcription accuracy. By default (`false`), low latency is not enabled. For more information, see Low latency.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Not available.
Next-generation models	Generally available or beta for next-generation models that support low latency. For more information, see Supported next-generation language models.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method

Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 16. The low_latency parameter

max_alternatives

An optional integer that specifies the maximum number of alternative hypotheses that the service returns. By default, the service returns a single final hypothesis. For more information, see Maximum alternatives.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 17. The max_alternatives parameter

model

An optional model that specifies the language in which the audio is spoken and the rate at which it was sampled: broadband/multimedia or narrowband/telephony. By default, `en-US_BroadbandModel` is used. For more information, see Using a model for speech recognition .

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Query parameter of `/v1/recognize` connection request
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 18. The model parameter

processing_metrics

An optional boolean that indicates whether the service returns metrics about its processing of the input audio. By default (`false`), the service returns no processing metrics. For more information, see Processing metrics.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available for all languages.
Next-generation models	Not available.

WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Not supported
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 19. The processing_metrics parameter

processing_metrics_interval

An optional float of at least 0.1 that indicates the interval at which the service is to return processing metrics. If the `processing_metrics` parameter is `true`, the service returns processing metrics every 1.0 seconds by default. For more information, see Processing metrics.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available for all languages.
Next-generation models	Not available.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Not supported
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 20. The processing_metrics_interval parameter

profanity_filter

An optional boolean that indicates whether the service censors profanity from a transcript. By default (`true`), profanity is filtered from the transcript. For more information, see Profanity filtering.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 21. The profanity_filter parameter

redaction

An optional boolean that indicates whether the service redacts numeric data with three or more consecutive digits from a transcript. By default (`false`), numeric data is not redacted. If you set the `redaction` parameter to `true`, the service automatically forces the `smart_formatting` parameter to be `true`, and it disables the `keywords`, `keywords_threshold`, `max_alternatives`, and (for the WebSocket interface) `interim_results` parameters. For more information, see Numeric redaction.

Availability and usage	Description

Large speech models	Beta for English and Japanese.
Previous-generation models	Beta for US English, Japanese, and Korean.
Next-generation models	Beta for US English, Japanese, and Korean.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 22. The redaction parameter

smart_formatting

An optional boolean that indicates whether the service converts dates, times, numbers, currency, and similar values into more conventional representations in the final transcript. For US English, the feature also converts certain keyword phrases into punctuation symbols. By default (`false`), smart formatting is not performed. For more information, see Smart formatting.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for US English, Japanese, and Spanish (all dialects).
Next-generation models	Generally available for US English, Japanese, and Spanish (all dialects). It also also available for the `en-WW_Medical_Telephony` model when US English audio is recognized.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 23. The smart_formatting parameter

smart_formatting_version

An optional boolean that indicates whether the service converts dates, times, numbers, currency, and similar values into more conventional representations in the final transcript. For more information, see Smart formatting version.

Availability and usage	Description
Large speech models	Generally available for all languages except Japanese.
Previous-generation models	Not supported.
Next-generation models	Generally available only for US English (including en-WW_Medical_Telephony), Brazilian Portuguese, France French, German, Canadian French and Spanish.
WebSocket	Parameter of JSON `start` message. The service disables interim results.

Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 23a. The smart_formatting_version parameter

speaker_labels

An optional boolean that indicates whether the service identifies which individuals spoke which words in a multi-participant exchange. If you set the `speaker_labels` parameter to `true`, the service automatically forces the `timestamps` parameter to be `true`. By default (`false`), speaker labels are not returned. For more information, see Speaker labels.

Availability and usage	Description
Large speech models	Beta for all languages.
Previous-generation models	Beta for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 24. The speaker_labels parameter

speech_detector_sensitivity

An optional float between 0.0 and 1.0 that indicates the sensitivity of speech recognition to non-speech events in the input audio. The default is 0.5, which provides a reasonable level of sensitivity to non-speech events. For more information, see Speech detector sensitivity.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all language models except for `ar-MS_BroadbandModel`, `pt-BR_BroadbandModel`, `zh-CN_BroadbandModel`, `zh-CN_NarrowbandModel`, and `de-DE_BroadbandModel`.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 25. The speech_detector_sensitivity parameter

split_transcript_at_phrase_end

An optional boolean that indicates whether the service splits a transcript into multiple final results based on semantic features of the input such as sentences. The service bases its understanding of semantic features on the base language model, which can be further influenced by custom language models and grammars. By default (`false`), the service produces no semantic splits. For more information, see Split transcript at phrase end.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 26. The split_transcript_at_phrase_end parameter

timestamps

An optional boolean that indicates whether the service produces timestamps for the words of the transcript. By default (`false`), timestamps are not returned. For more information, see [Word timestamps](#).

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 27. The timestamps parameter

Transfer-Encoding

An optional value of `chunked` that causes the audio to be streamed to the service. By default, audio is sent all at once as a one-shot delivery. For more information, see [Audio transmission](#).

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Not applicable; always streamed
Synchronous HTTP	Request header of `POST /v1/recognize` method
Asynchronous HTTP	Request header of `POST /v1/recognitions` method

Table 28. The Transfer-Encoding parameter

word_alternatives_threshold

An optional double between 0.0 and 1.0 that specifies the threshold at which the service reports acoustically similar alternatives for words of the input audio. By default, word alternatives are not returned. For more information, see Word alternatives.

Availability and usage	Description
Large speech models	Not available.
Previous-generation models	Generally available for all languages.
Next-generation models	Not available.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 29. The word_alternatives_threshold parameter

word_confidence

An optional boolean that indicates whether the service provides confidence measures for the words of the transcript. By default (`false`), word confidence measures are not returned. For more information, see Word confidence.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	Parameter of JSON `start` message
Synchronous HTTP	Query parameter of `POST /v1/recognize` method
Asynchronous HTTP	Query parameter of `POST /v1/recognitions` method

Table 30. The word_confidence parameter

X-Watson-Learning-Opt-Out

IBM Cloud

An optional boolean that indicates whether you opt out of the default request logging that IBM performs to improve the service for future users. To prevent IBM from accessing your data for general service improvements, specify `true` for the parameter. If you opt out, the service logs *no* user data from your requests, saving no audio or text to disk. You can also opt out at the account level. For more information, see Request logging.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	`x-watson-learning-opt-out` query parameter of `/v1/recognize` connection request
Synchronous HTTP	Request header of each request

Asynchronous HTTP	Request header of each request

Table 31. The X-Watson-Learning-Opt-Out parameter

X-Watson-Metadata

An optional string that associates a customer ID with data that is passed for recognition requests. The parameter accepts the argument `customer_id={id}`. By default, no customer ID is associated with the data. For more information, see Information security.

Availability and usage	Description
Large speech models	Generally available for all languages.
Previous-generation models	Generally available for all languages.
Next-generation models	Generally available for all languages.
WebSocket	`x-watson-metadata` query parameter of `/v1/recognize` connection request. (You must URL-encode the argument, for example, `customer_id%3dmy_customer_ID`.)
Synchronous HTTP	Request header of POST `/v1/recognize` request
Asynchronous HTTP	Request header of `POST /v1/register_callback` and `POST /v1/recognitions` requests

Table 32. The X-Watson-Metadata parameter

Customizing the service

Understanding customization

The IBM Watson® Speech to Text service offers a customization interface that you can use to augment its speech recognition capabilities. You can use customization to improve the accuracy of speech recognition requests by customizing a base model for your domain and audio.

The customization interface supports both custom language models and custom acoustic models. The interfaces for both types of custom model are similar and straightforward to use. Using either type of custom model with a recognition request is also straightforward: You specify the customization ID of the model with the request.

Speech recognition works the same with or without a custom model. When you use a custom model for speech recognition, you can use all of the parameters that are normally available with a recognition request. For more information about all available parameters, see the Parameter summary.

IBM Cloud You must have the Plus, Standard, or Premium pricing plan to use language model or acoustic model customization. Users of the Lite plan cannot use the customization interface, but they can upgrade to the Plus plan to gain access to customization. For more information, see the Pricing FAQs.

Language model customization

Note: For more information about the languages and models that support language model customization and their level of support (generally available or beta), see Language support for customization.

The service was developed with a broad, general audience in mind. The service's base vocabulary contains many words that are used in everyday conversation. Its models provide sufficiently accurate recognition for many applications. But they can lack knowledge of specific terms that are associated with particular domains.

The *language model customization* interface can improve the accuracy of speech recognition for domains such as medicine, law, information technology, and others. By using language model customization, you can expand and tailor the vocabulary of a base model to include domain-specific terminology.

You create a custom language model and add corpora and words specific to your domain. Once you train the custom language model on your enhanced vocabulary, you can use it for customized speech recognition. The service can typically train any custom model in a matter of minutes. The time and effort that are needed to create a custom model depend on the data that you have available for the model.

Language model customization is available for large speech models, previous-generation and next-generation models, but customization works differently for large speech models, previous- and next-generation models. The documentation describes the differences. For more information about getting started with language model customization, see

- Creating a custom language model
- Using a custom language model for speech recognition

Improved language model customization for next-generation models

Language model customization for next-generation models is being enhanced. The improvements to language model customization are being applied to next-generation models incrementally, starting with the models for a few languages. Over time, other next-generation language models will be migrated to the improved technology.

- *To identify models that use the improved technology,* look for a date in the *Language model customization (improved)* column of table 2 in Customization support for next-generation models. Check for a date for the version of the service that you use, IBM Cloud or IBM Cloud Pak for Data . That date indicates when the next-generation model was migrated to the improved technology.
- *To take advantage of the improved technology,* you must upgrade any custom language models that are based on an improved next-generation model. Once a custom model is based on an improved next-generation model, the service continues to perform any necessary upgrades when the custom model is retrained. For more information, see Upgrading a custom language model based on an improved next-generation model .

For language models that are based on the new technology, the following parameters have been optimized for improved speech recognition:

- The default `customization_weight` has changed from `0.2` to `0.1`. A non-default `customization_weight` that you had previously associated with your custom models is removed. For more information, see Using customization weight.
- The default `character_insertion_bias` remains `0.0`, but the models have changed in a manner that makes use of the parameter for speech recognition less necessary. For more information, see Character insertion bias.

Use the following guidelines when working with these parameters:

- If you use the default values for these parameters, continue to do so. The default values will likely continue to offer the best results for speech recognition.
- If you specify non-default values for these parameters, experiment with the default values. Your custom model might work well for speech

recognition with the default values.

- If you feel that using different values for these parameters might improve speech recognition with your custom model, experiment with incremental changes to determine whether the parameters are needed to improve speech recognition.

Supported next-generation language models:

- RNNT_CUSTOMIZATION_SUPPORTED_LANGS =
  - en-US_Multimedia
  - en-GB_Multimedia
  - en-AU_Multimedia
  - en-IN_Multimedia
  - en-US_Telephony
  - en-GB_Telephony
  - en-AU_Telephony
  - en-IN_Telephony
  - ja-JP_Multimedia
  - ja-JP_Telephony
  - fr-FR_Multimedia
  - fr-FR_Telephony
  - de-DE_Multimedia
  - de-DE_Telephony
  - fr-CA_Multimedia
  - pt-BR_Multimedia
  - pt-BR_Telephony

Acoustic model customization

Note: Acoustic model customization is available only for previous-generation models. It is not available for large speech models and next-generation models.

The service was developed with base acoustic models that work well for various audio characteristics. But in cases like the following, adapting a base model to suit your audio can improve speech recognition:

- Your acoustic channel environment is unique. For example, the environment is noisy, microphone quality or positioning are suboptimal, or the audio suffers from far-field effects.
- Your speakers' speech patterns are atypical. For example, a speaker talks abnormally fast or the audio includes casual conversations.
- Your speakers' accents are pronounced. For example, the audio includes speakers who are talking in a non-native or second language.

The *acoustic model customization* interface can adapt a base model to your environment and speakers. You create a custom acoustic model and add audio data (audio resources) that closely match the acoustic signature of the audio that you want to transcribe. Once you train the custom acoustic model with your audio resources, you can use it for customized speech recognition.

The length of time that it takes the service to train the custom model depends on how much audio data the model contains. In general, training takes twice the length of the cumulative audio. The time and effort that are needed to create a custom model depend on the audio data that you have available for the model. They also on whether you use transcriptions of the audio.

For more information about getting started with acoustic model customization, see

- Creating a custom acoustic model
- Using a custom acoustic model for speech recognition

Grammars

Note: For more information about the languages and models that support grammars and their level of support (generally available or beta), see Language support for customization.

Custom language models allow you to expand the service's base vocabulary. *Grammars* enable you to restrict the words that the service can recognize from that vocabulary. When you use a grammar with a custom language model for speech recognition, the service can recognize only words, phrases, and strings that are recognized by the grammar. Because the grammar defines a limited search space for valid matches, the service can deliver results faster and more accurately.

You add a grammar to a custom language model and train the model just as you do for a corpus. Unlike a corpus, however, you must explicitly specify that a grammar is to be used with a custom model during speech recognition.

Note: *Grammars* is available only for previous- and next-generation models. It is not available for large speech models.

For more information about getting started with grammars, see

- Using grammars with custom language models
- Understanding grammars
- Adding a grammar to a custom language model
- Using a grammar for speech recognition

Using acoustic and language customization together

Note: Using acoustic and language customization together applies only to previous-generation models. Also, some previous-generation models do not support both language and acoustic customization.

Using a custom acoustic model alone can improve the service's recognition capabilities. But if transcriptions or related corpora are available for your example audio, you can use that data to further improve the quality of speech recognition based on the custom acoustic model.

By creating a custom language model that complements your custom acoustic model, you can enhance speech recognition by using the two models together. When you train a custom acoustic model, you can specify a custom language model that includes transcriptions of the audio resources or a vocabulary of domain-specific words from the resources. Similarly, when you transcribe audio, the service accepts a custom language model, a custom acoustic model, or both. And if your custom language model includes a grammar, you can use that model and grammar with a custom acoustic model for speech recognition.

For more information, see Using custom acoustic and custom language models together

Upgrading custom models

To improve the quality of speech recognition, the service occasionally updates base large speech models, previous- and next-generation models. An update to a base model affects only that model. The update does not affect any other models of the same or different languages.

An update to a base model can require that you upgrade any custom models that are built on that base model to take advantage of the improvements. An update to a base model that requires an upgrade produces a new version of the base model. An update that does not require an upgrade does not produce a new version.

- *For previous-generation models,* all updates to a base model produce a new version of the model. When a new version of a base model is released, you must upgrade any custom language and custom acoustic models that are built on the updated base model to take advantage of the improvements.
- *For large speech models and next-generation models,* most updates do not produce a new version of the model. These updates do not require that you upgrade your custom models. Some updates, however, do generate a new version of a base model. You must upgrade any custom language models that are built on the updated base model to take advantage of the improvements.

All updates to base models and whether they require upgrades are announced in the release notes:

- Release notes for Speech to Text for IBM Cloud
- Release notes for Speech to Text for IBM Cloud Pak for Data

Once you upgrade a custom model, the service uses the latest version of the custom model by default when you specify that model with a speech recognition request. But you can still direct the service to use the older version of the model. For more information about upgrading custom models and about using an older version of a model, see

- Upgrading custom models
- Using upgraded custom models for speech recognition

Language support for customization

The IBM Watson® Speech to Text service supports language model customization, acoustic model customization, and grammars at different levels for different languages and types of models. The tables in the following sections document the service's level of support for all languages and models:

- *GA* indicates that the feature is generally available for production use for that model.

- *Beta* indicates that the feature is available as a beta offering for that model.
- *Not supported* means that the feature is not available for that model.

Where applicable, customization features are also identified as specific to IBM Cloud® or IBM Cloud Pak® for Data. Unless otherwise indicated, a feature is available for both versions of the service.

Customization support for previous-generation models

Starting **August 1, 2023**, all previous-generation models are now **discontinued** from the service. New clients must now only use the next-generation models. All existing clients must now migrate to the equivalent next-generation model. For more information, see Migrating to next-generation models.

Table 1 lists the previous-generation models that are supported for language model customization, grammars, and acoustic model customization. To learn which models are supported for speech recognition for IBM Cloud®, IBM Cloud Pak® for Data, or both, see Supported previous-generation language models.

Language (dialect)	Models	Language model customization	Grammars	Acoustic model customization
Arabic (Modern Standard)	`ar-MS_BroadbandModel` Discontinued	Not supported	Not supported	GA
Chinese (Mandarin)	`zh-CN_NarrowbandModel` Discontinued	Not supported	Not supported	GA
	`zh-CN_BroadbandModel` Discontinued	Not supported	Not supported	GA
Dutch (Netherlands)	`nl-NL_NarrowbandModel` Discontinued	GA	GA	GA
	`nl-NL_BroadbandModel` Discontinued	GA	GA	GA
English (Australian)	`en-AU_NarrowbandModel` Discontinued	GA	GA	GA
	`en-AU_BroadbandModel` Discontinued	GA	GA	GA
English (United Kingdom)	`en-GB_NarrowbandModel` Discontinued	GA	GA	GA
	`en-GB_BroadbandModel` Discontinued	GA	GA	GA
English (United States)	`en-US_NarrowbandModel` Discontinued	GA	GA	GA
	`en-US_BroadbandModel` Discontinued	GA	GA	GA
	`en-US_ShortForm_NarrowbandModel` Discontinued	GA	GA	GA
French (Canadian)	`fr-CA_NarrowbandModel` Discontinued	GA	GA	GA
	`fr-CA_BroadbandModel` Discontinued	GA	GA	GA

French (France)	`fr-FR_NarrowbandModel` Discontinued	GA	GA	GA
	`fr-FR_BroadbandModel` Discontinued	GA	GA	GA
German	`de-DE_NarrowbandModel` Discontinued	GA	GA	GA
	`de-DE_BroadbandModel` Discontinued	GA	GA	GA
Italian	`it-IT_NarrowbandModel` Discontinued	GA	GA	GA
	`it-IT_BroadbandModel` Discontinued	GA	GA	GA
Japanese	`ja-JP_NarrowbandModel` Discontinued	GA	GA	GA
	`ja-JP_BroadbandModel` Discontinued	GA	GA	GA
Korean	`ko-KR_NarrowbandModel` Discontinued	GA	GA	GA
	`ko-KR_BroadbandModel` Discontinued	GA	GA	GA
Portuguese (Brazilian)	`pt-BR_NarrowbandModel` Discontinued	GA	GA	GA
	`pt-BR_BroadbandModel` Discontinued	GA	GA	GA
Spanish (Argentinian)	`es-AR_NarrowbandModel` Discontinued	Beta	Beta	Beta
	`es-AR_BroadbandModel` Discontinued	Beta	Beta	Beta
Spanish (Castilian)	`es-ES_NarrowbandModel` Discontinued	GA	GA	GA
	`es-ES_BroadbandModel` Discontinued	GA	GA	GA
Spanish (Chilean)	`es-CL_NarrowbandModel` Discontinued	Beta	Beta	Beta
	`es-CL_BroadbandModel` Discontinued	Beta	Beta	Beta
Spanish (Colombian)	`es-CO_NarrowbandModel` Discontinued	Beta	Beta	Beta
	`es-CO_BroadbandModel` Discontinued	Beta	Beta	Beta

Spanish (Mexican)	`es-MX_NarrowbandModel` Discontinued	Beta	Beta	Beta
	`es-MX_BroadbandModel` Discontinued	Beta	Beta	Beta
Spanish (Peruvian)	`es-PE_NarrowbandModel` Discontinued	Beta	Beta	Beta
	`es-PE_BroadbandModel` Discontinued	Beta	Beta	Beta

Table 1. Previous-generation language support for customization

Customization support for next-generation models

Table 2 lists the next-generation models that are supported for language model customization and grammars. Next-generation language models do not support acoustic model customization. To learn which models are supported for speech recognition for IBM Cloud®, IBM Cloud Pak® for Data, or both, see Supported next-generation language models.

Note: In the *Language model customization* column, an *(improved)* date indicates when a model was migrated to the new language model customization technology. Make sure to check for a date for the version of the service that you use, *IBM Cloud* or *IBM Cloud Pak for Data* . Models that do not include an *(improved)* date are not yet migrated to the new technology. For more information about the improved technology, see Improved language model customization for next-generation models .

Language (dialect)	Models	Language model customization (improved)	Grammars
Arabic (Modern Standard)	`ar-MS_Telephony`	GA	GA
Chinese (Mandarin)	`zh-CN_Telephony`	GA	GA
Czech	`cs-CZ_Telephony`	GA	GA
Dutch (Belgian)	`nl-BE_Telephony`	GA	GA
Dutch (Netherlands)	`nl-NL_Telephony`	GA	GA
	`nl-NL_Multimedia`	GA	GA
English (Australian)	`en-AU_Telephony`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA
	`en-AU_Multimedia`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA
English (Indian)	`en-IN_Telephony`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA
English (United Kingdom)	`en-GB_Telephony`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA

	`en-GB_Multimedia`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA
English (United States)	`en-US_Telephony`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA
	`en-US_Multimedia`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA
English (all supported dialects)	`en-WW_Medical_Telephony`	Beta	Beta
French (Canadian)	`fr-CA_Telephony`	GA	GA
	`fr-CA_Multimedia`	GA	GA
French (France)	`fr-FR_Telephony`	GA	GA
	`fr-FR_Multimedia`	GA	GA
German	`de-DE_Telephony`	GA	GA
	`de-DE_Multimedia`	GA	GA
Hindi	`hi-IN_Telephony`	GA	GA
Italian	`it-IT_Telephony`	GA	GA
	`it-IT_Multimedia`	GA	GA
Japanese	`ja-JP_Telephony`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA
	`ja-JP_Multimedia`	GA IBM Cloud 27 February 2023 IBM Cloud Pak for Data 1 May 2023	GA
Korean	`ko-KR_Telephony`	GA	GA
	`ko-KR_Multimedia`	GA	GA
Portuguese (Brazilian)	`pt-BR_Telephony`	GA	GA
	`pt-BR_Multimedia`	GA	GA
Spanish (Castilian)	`es-ES_Telephony`	GA	GA
	`es-ES_Multimedia`	GA	GA

Spanish (Argentinian, Chilean, Colombian, Mexican, and Peruvian)	`es-LA_Telephony`	GA	GA
Swedish	`sv-SE_Telephony`	GA	GA

Table 2. Next-generation language support for customization

Customization support for large speech models

Table 3 lists the large speech models that are supported for language model customization and grammars. Large speech models do not support acoustic model customization. To learn which models are supported for speech recognition for IBM Cloud®, IBM Cloud Pak® for Data, or both, see Supported large speech languages and models.

Note: Make sure to check for a date for the version of the service that you use, IBM Cloud or IBM Cloud Pak for Data.

Language (dialect)	Models	Language model customization	Grammars
English (Australian)	`en-AU`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024	N/A
English (Indian)	`en-IN`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024	N/A
English (United Kingdom)	`en-GB`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024	N/A
English (United States)	`en-US`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024	N/A
French (Canadian)	`fr-CA`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024	N/A
French (France)	`fr-FR`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024	N/A
Japanese	`ja-JP`	IBM Cloud 20 May 2024 IBM Cloud Pak for Data 12 June 2024	N/A
Portuguese (Brazilian)	`pt-BR`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024	N/A
Portuguese (Portugal)	`pt-PT`	IBM Cloud 23 August 2024 IBM Cloud Pak for Data 23 August 2024	N/A
Spanish (Castilian)	`es-ES`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024	N/A
Spanish (Argentinian)	`es-AR`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024	N/A
Spanish (Chilean)	`es-CL`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024	N/A

Spanish (Colombian)	`es-CO`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024	N/A
Spanish (Mexican)	`es-MX`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024	N/A
Spanish (Peruvian)	`es-PE`	IBM Cloud 18 June 2024 IBM Cloud Pak for Data 23 August 2024	N/A

Table 3. Large speech models language support for customization

Usage notes for customization

To use customization with the IBM Watson® Speech to Text service, you need to be aware of topics such as custom model ownership, limits on the number of custom models that you can create, and information security and request logging. These topics are common to all supported languages and model types.

Ownership of custom models

A custom model is owned by the instance of the Speech to Text service whose credentials are used to create it. To work with the custom model in any way, you must use credentials for that instance of the service with methods of the customization interface. Credentials that are created for other instances of the service cannot view or access the custom model.

All credentials that are obtained for the same instance of the Speech to Text service share access to all custom models created for that service instance. To restrict access to a custom model, create a separate instance of the service and use only the credentials for that service instance to create and work with the model. Credentials for other service instances cannot affect the custom model.

An advantage of sharing ownership across credentials for a service instance is that you can cancel a set of credentials, for example, if they become compromised. You can then create new credentials for the same service instance and still maintain ownership of and access to custom models created with the original credentials.

Maximum number of custom models

You can create no more than 1024 custom language models and no more than 1024 custom acoustic models per owning service credentials. If you try to create more than 1024 custom models of either type, the service returns an error. You do not lose any existing models, but you cannot create any more until your model count is below the limit of 1024 for the type of custom model that you are trying to create.

Information security

You can associate a customer ID with data that is added or updated for custom language and custom acoustic models. You associate a customer ID with corpora, custom words, grammars, and audio resources by passing the `X-Watson-Metadata` header with the following methods. If necessary, you can then delete the data by using the `DELETE /v1/user_data` method.

- `POST /v1/customizations/{customization_id}/corpora/{corpus_name}`
- `POST /v1/customizations/{customization_id}/words`
- `PUT /v1/customizations/{customization_id}/words/{word_name}`
- `POST /v1/customizations/{customization_id}/grammars/{grammar_name}`
- `POST /v1/acoustic_customizations/{customization_id}/audio/{audio_name}`

In addition, if you delete an instance of the Speech to Text service from the IBM Cloud console, all data associated with that service instance is automatically deleted. This includes all custom language models, corpora, grammars, and words, and all custom acoustic models and audio resources. This data is purged automatically and regardless of whether a customer ID is associated with the data.

For more information, see Information security.

Request logging and data privacy

IBM Cloud

How the service handles request logging for calls to the customization interface depends on the request:

- The service *does not* log data that is used to build custom models. For example, when working with corpora and words in a custom language model, you do not need to set the `X-Watson-Learning-Opt-Out` request header. Your training data is never used to improve the service's base models.

- The service *does* log data when a custom model is used with a recognition request. You can opt out of request logging at the account level or by setting the `X-Watson-Learning-Opt-Out` request header to `true`.

For more information, see Request logging.

Language model customization

Creating a custom language model

Follow these steps to create, add contents to, and train a custom language model for the IBM Watson® Speech to Text service:

1. Create a custom language model. You can create multiple custom models for the same or different domains. The process is the same for any model that you create. Language model customization is available for all large speech models, most previous-generation models and for all next-generation models. For more information, see Language support for customization.
2. Add a corpus to the custom language model. A corpus is a plain text document that uses terminology from the domain in context. You can add multiple corpora serially, one at a time, to a custom model. *For custom models that are based on previous-generation models,* the service builds a vocabulary for a custom model by extracting terms from corpora that do not exist in its base vocabulary. *For custom models that are based on next-generation models,* the service extracts character sequences rather than words from corpora.
3. Add words to the custom language model. You can also add custom words to a model individually. You can specify how the words from a custom model are to be displayed in a speech transcript and how they are pronounced in audio. *For custom models that are based on previous-generation models,* you can also modify custom words that are extracted from corpora.
4. Train the custom language model. After you add corpora and words to the custom model, you must train the model. Training prepares the custom model for use in speech recognition. The model does not use new or modified corpora or words until you train it.
5. Use a custom language model for speech recognition. After you train your custom model, you can use it with speech recognition requests. If the audio that is passed for transcription contains domain-specific words that are defined in the custom model's corpora and custom words, the results of the request reflect the model's enhanced vocabulary. You can use only one model at a time with a speech recognition request.

The steps for creating a custom language model are iterative. You can add corpora, add words, and train or retrain a model as often as needed. You can also add grammars to most custom language model. Grammars restrict the service's response to only those words that are recognized by a grammar.

- For more information about using grammars, see Using grammars with custom language models.
- For more information about the languages and models that support grammars, see Language support for customization.

Create a custom language model

You use the `POST /v1/customizations` method to create a new custom language model. The method accepts a JSON object that defines the attributes of the new custom model as the body of the request. The new custom model is owned by the instance of the service whose credentials are used to create it. For more information, see Ownership of custom models.

You can create a maximum of 1024 custom language models per owning credentials. For more information, see Maximum number of custom models.

A new custom language model has the following attributes:

`name` (*required* string)

A user-defined name for the new custom model. Use a localized name that matches the language of the custom model and describes the domain of the model, such as `Medical custom model` or `Legal custom model`.

- Include a maximum of 256 characters in the name.
- Do not use backslashes, slashes, colons, equal signs, ampersands, or question marks in the name.
- Use a name that is unique among all custom language models that you own.

`base_model_name` (*required* string)

The name of the base language model that is to be customized by the new custom model. You must use the name of a model that is returned by the `GET /v1/models` method. The new custom model can be used only with the base model that it customizes.

`dialect` (*optional* string)

The dialect of the specified language that is to be used with the new custom model. *For all languages, it is always safe to omit this field.* The service automatically uses the language identifier from the name of the base model. For example, the service automatically uses `en-US` for all US English models.

If you specify the `dialect` for a new custom model, follow these guidelines:

- For non-Spanish previous-generation models and for next-generation models, you must specify a value that matches the five-character language identifier from the name of the base model.
- For Spanish previous-generation models, you must specify one of the following values:
  - `es-ES` for Castilian Spanish (`es-ES` models)
  - `es-LA` for Latin American Spanish (`es-AR` , `es-CL` , `es-CO` , and `es-PE` models)
  - `es-US` for Mexican (North American) Spanish (`es-MX` models)

All values that you pass for the `dialect` field are case-insensitive.

`description` (*optional* string)

A recommended description of the new custom model.

- Use a localized description that matches the language of the custom model.
- Include a maximum of 128 characters in the description.

The following example creates a new custom language model named `Example model` . The model is created for the base model `en-US-BroadbandModel` and has the description `Example custom language model` . The required `Content-Type` header specifies that JSON data is being passed to the method.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--data "{\"name\": \"Example model\", \
  \"base_model_name\": \"en-US_BroadbandModel\", \
  \"description\": \"Example custom language model\"}" \
"{url}/v1/customizations"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--data "{\"name\": \"Example model\", \
  \"base_model_name\": \"en-US_BroadbandModel\", \
  \"description\": \"Example custom language model\"}" \
"{url}/v1/customizations"
```

The example returns the customization ID of the new model. Each custom model is identified by a unique customization ID, which is a Globally Unique Identifier (GUID). You specify a custom model's GUID with the `customization_id` parameter of calls that are associated with the model.

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96"
}
```

Add a corpus to the custom language model

Once you create your custom language model, the next step is to add domain-specific data to the model. The recommended means of populating a custom model is to add one or more corpora. A corpus is a plain text file that ideally contains sample sentences from your domain.

- *For custom models that are based on large speech models,* the service parses and extracts word sequences from one or multiple corpora files. The characters help the service learn and predict character sequences from audio. For more information about using corpora with custom models that are based on large speech models, see [Working with corpora for large speech models and next-generation models]().
- *For custom models that are based on previous-generation models,* the service parses a corpus file and extracts any words that are not in its base vocabulary. Such words are referred to out-of-vocabulary (OOV) words. For more information about using corpora with custom models that are based on previous-generation models, see [Working with corpora for previous-generation models]().
- *For custom models that are based on next-generation models,* the service parses and extracts character sequences from a corpus file. The characters help the service learn and predict character sequences from audio. For more information about using corpora with custom models that are based on next-generation models, see [Working with corpora for large speech models and next-generation models]().

Tip: By providing sentences that include domain-specific words, corpora allow the service to learn the words and character sequences in context. You can also augment and modify a model's words individually. Training a model only on individual words as opposed to words added from corpora is more time-consuming and can produce less effective results.

You use the `POST /v1/customizations/{customization_id}/corpora/{corpus_name}` method to add a corpus to a custom model:

`customization_id` (*required* string)

Specify the customization ID of the custom model to which the corpus is to be added.

`corpus_name` (*required* string)

Specify a name for the corpus. Use a localized name that matches the language of the custom model and reflects the contents of the corpus.

- Include a maximum of 128 characters in the name.
- Do not use characters that need to be URL-encoded. For example, do not use spaces, slashes, backslashes, colons, ampersands, double quotes, plus signs, equals signs, questions marks, and so on in the name. (The service does not prevent the use of these characters. But because they must be URL-encoded wherever used, their use is strongly discouraged.)
- Do not use the name of a corpus or grammar that has already been added to the custom model.
- Do not use the name `user`, which is reserved by the service to denote custom words that are added or modified by the user.
- Do not use the name `base_lm` or `default_lm`. Both names are reserved for future use by the service.

Pass the corpus text file as the required body of the request. The following example adds the corpus text file `healthcare.txt` to the custom model with the specified ID. The example names the corpus `healthcare`.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--data-binary @healthcare.txt \
"{url}/v1/customizations/{customization_id}/corpora/healthcare"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--data-binary @healthcare.txt \
"{url}/v1/customizations/{customization_id}/corpora/healthcare"
```

The method also accepts an optional `allow_overwrite` query parameter that overwrites an existing corpus for a custom model. Use the parameter if you need to update a corpus file after you add it to a model.

The method is asynchronous. It can take on the order of minutes to complete. The duration of the operation depends on the total number of words in the corpus and the current load on the service. *For custom models that are based on previous-generation models,* the duration also depends on the number of new words that the service finds in the corpus. For more information about checking the status of a corpus, see Monitoring the add corpus request.

You can add any number of corpora to a custom model by calling the method once for each corpus text file. The addition of one corpus must be fully complete before you can add another. A corpus has the status `being_processed` when you first add it to a model. Its status becomes `analyzed` when the service finishes processing it.

For custom models that are based on previous-generation models, after the addition of a corpus is complete, examine the new custom words that were extracted from it to check for typographical and other errors. For more information, see Validating a words resource for previous-generation models.

Monitoring the add corpus request

The service returns a 201 response code if the corpus is valid. It then asynchronously processes the contents of the corpus. You cannot submit requests to add data to the custom model or to train the model until the service's analysis of the corpus for the current request completes.

To determine the status of the analysis, use the `GET /v1/customizations/{customization_id}/corpora/{corpus_name}` method to poll the status of the corpus. The method accepts the ID of the model and the name of the corpus, as shown in the following example:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/corpora/corpus1"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/corpora/corpus1"
```

The response includes the status of the corpus. Because the custom model is based on a previous-generation model, the response shows the number of OOV words.

```
{
  "name": "corpus1",
  "total_words": 5037,
  "out_of_vocabulary_words": 401,
  "status": "analyzed"
}
```

The `status` field has one of the following values:

- `analyzed` indicates that the service has successfully analyzed the corpus.
- `being_processed` indicates that the service is still analyzing the corpus.
- `undetermined` indicates that the service encountered an error while processing the corpus.

Use a loop to check the status of the corpus every 10 seconds until it becomes `analyzed`. For more information about checking the status of a model's corpora, see [Listing corpora for a custom language model]().

Add words to the custom language model

Although adding corpora is the recommended means of adding words to a custom language model, you can also add individual custom words to the model directly. The service parses custom words for the custom model just as it does the contents of words from corpora.

If you have only one or a few words to add to a model, using corpora to add the words might not be practical or even viable. The simplest approach is to add a word with only its spelling. But you can also indicate how the word is to be displayed and one or more pronunciations for the word.

- For more information about adding words to a custom model that is based on a *previous-generation model*, see Working with custom words for previous-generation models.
- For more information about adding words to a custom model that is based on *large speech models and next-generation models*, see Working with custom words for large speech models and next-generation models.

After you add words to a custom model, examine the new custom words to check for typographical and other errors. This check is especially important when you add multiple words at one time.

- *For custom models that are based on previous-generation models,* see Validating a words resource for previous-generation models.
- *For custom models that are based on large speech models and next-generation models,* see Validating a words resource for large speech models and next-generation models.

Add words with the POST method

The `POST /v1/customizations/{customization_id}/words` method adds one or more words at one time. You pass the words to be added as JSON data via the body of the request or from a file. In either case, the required `Content-Type` header specifies that JSON data is being passed to the method.

The following examples add two custom words, `HHonors` and `IEEE`, to the custom model with the specified ID:

- The first example passes the information about each word via the body of the request:

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  --header "Content-Type: application/json" \
  --data "{\"words\": [ \
     {\"word\": \"HHonors\", \"sounds_like\": [\"hilton honors\", \"H. honors\"], \"display_as\": \"HHonors\"}, \
     {\"word\": \"IEEE\", \"sounds_like\": [\"I. triple E.\"]}]}" \
  "{url}/v1/customizations/{customization_id}/words"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X POST \
  --header "Authorization: Bearer {token}" \
  --header "Content-Type: application/json" \
  ```

```
--data "{\"words\": [ \
  {\"word\": \"HHonors\", \"sounds_like\": [\"hilton honors\", \"H. honors\"], \"display_as\": \"HHonors\"}, \
  {\"word\": \"IEEE\", \"sounds_like\": [\"I. triple E.\"]}]}" \
"{url}/v1/customizations/{customization_id}/words"
```

- The second example adds the same words from a file named `words.json`:

  ```
  {
    "words": [
      {"word": "HHonors", "sounds_like": ["hilton honors", "H. honors"], "display_as": "HHonors"},
      {"word": "IEEE", "sounds_like": ["I. triple E."]}
    ]
  }
  ```

  The following request adds the words from the file:

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  --header "Content-Type: application/json" \
  --data-binary @words.json \
  "{url}/v1/customizations/{customization_id}/words"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X POST \
  --header "Authorization: Bearer {token}" \
  --header "Content-Type: application/json" \
  --data-binary @words.json \
  "{url}/v1/customizations/{customization_id}/words"
  ```

The `POST` method is asynchronous. It can take on the order of minutes to complete. The time that it takes to complete depends on the number of words that you add and the current load on the service. For more information about checking the status of the operation, see Monitoring the add words request.

Add a word with the PUT method

The `PUT /v1/customizations/{customization_id}/words/{word_name}` method adds individual words. You pass a JSON object that provides information about the word as the body of the request.

The following example adds the word `NCAA` to the model with the specified ID. The required `Content-Type` header again indicates that JSON data is being passed to the method.

IBM Cloud

```
$ curl -X PUT -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--data "{\"sounds_like\": [\"N. C. A. A.\", \"N. C. double A.\"]}" \
"{url}/v1/customizations/{customization_id}/words/NCAA"
```

IBM Cloud Pak for Data

```
$ curl -X PUT \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--data "{\"sounds_like\": [\"N. C. A. A.\", \"N. C. double A.\"]}" \
"{url}/v1/customizations/{customization_id}/words/NCAA"
```

The `PUT` method is synchronous. The service returns a response code that reports the success or failure of the request immediately.

Monitoring the add words request

When you use the `POST /v1/customizations/{customization_id}/words` method, the service returns a 201 response code if the input data is valid. It then asynchronously processes the words to add them to the model. You cannot submit requests to add data to the custom model or to train the model until the service completes the request to add new words.

To determine the status of the request, use the `GET /v1/customizations/{customization_id}` method to poll the model's status. The method accepts the customization ID of the model, as in the following example:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}"
```

The request includes information about the status of the model:

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96",
  "created": "2016-06-01T18:42:25.324Z",
  "updated": "2016-06-01T18:45:11.737Z",
  "language": "en-US",
  "dialect": "en-US",
  "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
  "name": "Example model",
  "description": "Example custom language model",
  "base_model_name": "en-US_BroadbandModel",
  "status": "pending",
  "progress": 0
}
```

The `status` field reports the current state of the model. While the service is processing new words, the status remains `pending`. Use a loop to check the status every 10 seconds until it becomes `ready` to indicate that the operation is complete. For more information about possible `status` values, see Monitoring the train model request.

Modifying words in a custom model

You can also use the `POST /v1/customizations/{customization_id}/words` and `PUT /v1/customizations/{customization_id}/words/{word_name}` methods to modify or augment a word in a custom model. You might need to use the methods to correct a typographical error or other mistake that was made when a word was added to the model. You might also need to add sounds-like definitions for an existing word.

You use the methods to modify the definition of an existing word exactly as you do to add a word. The new data that you provide for the word overwrites the word's existing definition. *For custom models that are based on previous-generation models,* you can also modify words that were added from corpora.

Train the custom language model

Once you populate a custom language model with new words (by adding corpora, by adding words directly, or by adding grammars), you must train the model on the new data. Training prepares the custom model to use the data in speech recognition. The model does not use words that you add via any means until you train it on the data.

You use the `POST /v1/customizations/{customization_id}/train` method to train a custom model. You pass the method the customization ID of the model that you want to train, as in the following example:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/train"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/train"
```

The method is asynchronous. Training can take on the order of minutes to complete depending on the number of new words on which the model is being trained and the current load on the service. For more information about checking the status of a training operation, see Monitoring the train model request.

The method includes the following optional query parameters:

- The `word_type_to_add` parameter specifies the words on which the custom model is to be trained:
  - Specify `all` or omit the parameter to train the model on all of its words, regardless of their origin.
  - Specify `user` to train the model only on words that were added or modified by the user, ignoring words that were extracted only from corpora or grammars.

  For custom models that are based on previous-generation models, this option is useful if you add corpora with noisy data, such as words that contain typographical errors. Before training the model on such data, you can use the `word_type` query parameter of the `GET /v1/customizations/{customization_id}/words` method to review words that are extracted from corpora and grammars. For more information, see [Listing custom words from a custom language model](#).

  For custom models that are based on large speech models and next-generation models, the service ignores the `word_type_to_add` parameter. The words resource contains only custom words that the user adds or modifies directly, so the parameter is unnecessary.

- The `customization_weight` parameter specifies the relative weight that is given to words from the custom model as opposed to words from the base vocabulary when the custom model is used for speech recognition. You can also specify a customization weight with any recognition request that uses the custom model. For more information, see [Using customization weight](#).

- The `strict` parameter indicates whether training is to proceed if the custom model contains a mix of valid and invalid resources (corpora, words, and grammars). By default, training fails if the model contains one or more invalid resources. Set the parameter to `false` to allow training to proceed as long as the model contains at least one valid resource. The service excludes invalid resources from the training. For more information, see [Training failures for custom language models](#).

Monitoring the train model request

The service returns a 200 response code if it initiates the training process successfully. The service cannot accept subsequent training requests, or requests to add new corpora, words, or grammars, until the existing request completes.

Note: Adding custom words directly to a custom model that is based on a large speech model or next-generation model, as described in [Add words to the custom language model](#), causes training of a model to take a few minutes longer than it otherwise would. If you are training a model with custom words that you added by using the `POST /v1/customizations/{customization_id}/words` or `PUT /v1/customizations/{customization_id}/words/{word_name}` method, allow for some minutes of extra training time for the model.

To determine the status of a training request, use the `GET /v1/customizations/{customization_id}` method to poll the model's status. The method accepts the customization ID of the model:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}"
```

The response includes information about the model's status:

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96",
  "created": "2016-06-01T18:42:25.324Z",
  "updated": "2016-06-01T18:45:11.737Z",
  "language": "en-US",
  "dialect": "en-US",
  "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
  "name": "Example model",
  "description": "Example custom language model",
  "base_model_name": "en-US_BroadbandModel",
  "status": "training",
  "progress": 0
}
```

The response includes `status` and `progress` fields that report the state of the custom model. The meaning of the `progress` field depends on the model's status. The `status` field can have one of the following values:

- `pending` indicates that the model was created but is waiting either for valid training data to be added or for the service to finish analyzing data that was added. The `progress` field is `0`.
- `ready` indicates that the model contains valid data and is ready to be trained. The `progress` field is `0`.

  If the model contains a mix of valid and invalid resources (for example, both valid and invalid custom words), training of the model fails unless you set the `strict` query parameter to `false`. For more information, see Training failures for custom language models.
- `training` indicates that the model is being trained. The `progress` field is `0`. The field changes from `0` to `100` when training is complete.
- `available` indicates that the model is trained and ready to use. The `progress` field is `100`.
- `upgrading` indicates that the model is being upgraded. The `progress` field is `0`.
- `failed` indicates that training of the model failed. The `progress` field is `0`. For more information, see Training failures for custom language models.

Use a loop to check the status every 10 seconds until it becomes `available`. For more information about checking the status of a custom model, see Listing custom language models.

Training failures for custom language models

Training fails to start if the service is handling another request for the custom language model. For instance, a training request fails to start with a status code of 409 if the service is

- Processing a corpus or grammar to generate a list of OOV words or to extract character sequences
- Processing custom words to validate or auto-generate sounds-like pronunciations
- Handling another training request

Training also fails to start with a status code of 400 if the custom model

- Contains no new valid training data (corpora, words, or grammars) since it was created or last trained
- Contains one or more invalid corpora, words, or grammars (for example, a custom word has an invalid sounds-like pronunciation)

If the training request fails with a status code of 400, the service sets the custom model's status to `failed`. Take one of the following actions:

- Use methods of the customization interface to examine the model's resources and fix any errors that you find:
  - For an invalid corpus, you can correct the corpus text file and use the `allow_overwrite` parameter of the `POST /v1/customizations/{customization_id}/corpora/{corpus_name}` method to add the corrected file to the model. For more information, see Add a corpus to the custom language model .
  - For an invalid grammar, you can correct the grammar file and use the `allow_overwrite` parameter of the `POST /v1/customizations/{customization_id}/grammars/{grammar_name}` method to add the corrected file to the model. For more information, see Add a grammar to the custom language model .
  - For an invalid custom word, you can use the `POST /v1/customizations/{customization_id}/words` or `PUT /v1/customizations/{customization_id}/words/{word_name}` method to modify the word directly in the model's words resource. For more information, see Modifying words in a custom model.

  For more information about validating the words in a custom language model, see

  - Validating a words resource for previous-generation models
  - Validating a words resource for large speech models and next-generation models .
- Set the `strict` parameter of the `POST /v1/customizations/{customization_id}/train` method to `false` to exclude invalid resources from the training. The model must contain at least one valid resource (corpus, word, or grammar) for training to succeed. The `strict` parameter is useful for training a custom model that contains a mix of valid and invalid resources.

Using a custom language model for speech recognition

Once you create and train your custom language model, you can use it in speech recognition requests by using the `language_customization_id` query parameter. By default, no custom language model is used with a request. You can create multiple custom language models for the same or different domains. But you can specify only one custom language model at a time for a speech recognition request. You must issue the request with credentials for the instance of the service that owns the custom model.

A custom model can be used only with the base model for which it is created. If your custom model is based on a model other than the default, you must also specify that base model with the `model` query parameter. For more information, see Using the default model.

For information about telling the service how much weight to give to words from a custom model, see Using customization weight. For examples that use a grammar with a custom language model, see Using a grammar for speech recognition .

Examples of using a custom language model

The following examples show the use of a custom language model with each speech recognition interface. In this case, the custom model that is used is based on the next-generation model `en-US_Telephony`.

- For the WebSocket interface, use the `/v1/recognize` method. The specified custom model is used for all requests that are sent over the connection.

```
var access_token = {access_token};
var wsURI = '{ws_url}/v1/recognize'
  + '?access_token=' + access_token
  + '&model=en-US_Telephony'
  + '&language_customization_id={customization_id}';
var websocket = new WebSocket(wsURI);
```

- For the synchronous HTTP interface, use the `POST /v1/recognize` method. The specified custom model is used for that request.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognize?model=en-US_Telephony&language_customization_id={customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognize?model=en-US_Telephony&language_customization_id={customization_id}"
```

- For the asynchronous HTTP interface, use the `POST /v1/recognitions` method. The specified custom model is used for that request.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognitions?model=en-US_Telephony&language_customization_id={customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognitions?model=en-US_Telephony&language_customization_id={customization_id}"
```

Using customization weight

A custom language model is a combination of the custom model and the base model that it customizes. You can tell the service how much weight to give to words from the custom language model compared to words from the base model for speech recognition. The weight that is assigned to a custom model is referred to as its *customization weight*.

You specify the relative weight for a custom language model as a double between 0.0 to 1.0:

- For custom models that are based on large speech models, the default customization weight is 0.5.
- For custom models that are based on previous-generation models, the default customization weight is 0.3.
- For custom models that are based on most next-generation models, the default customization weight is 0.2.
- For custom models that are based on improved next-generation models, the default customization weight is 0.1.

Tip: To identify models that use improved language model customization, look for an *(improved)* date in the *Language model customization* column of table 2 in Customization support for next-generation models. Make sure to check for a date for the version of the service that you use, *IBM Cloud* or *IBM Cloud Pak for Data* .

The default weight yields the best performance in the general case. It allows both words from the custom model and words from the base vocabulary to be recognized.

However, in cases where the audio to be transcribed makes frequent use of words from the custom model, increasing the customization weight can improve the accuracy of transcription results. Exercise caution when you set the customization weight. Although a higher weight can improve the accuracy of phrases from the domain of the custom model, it can also negatively impact performance on non-domain phrases. (Even if you set the weight to 0.0, a small probability exists that the transcription can include custom words due to the implementation of language model customization.)

You specify a customization weight by using the `customization_weight` parameter. You can specify the parameter when you train a custom language model or when you use it with a speech recognition request.

- For a training request, the following example specifies a customization weight of `0.5` with the `POST /v1/customizations/{customization_id}/train` method:

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  "{url}/v1/customizations/{customization_id}/train?customization_weight=0.5"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X POST \
  --header "Authorization: Bearer {token}" \
  "{url}/v1/customizations/{customization_id}/train?customization_weight=0.5"
  ```

  Setting a customization weight during training saves the weight with the custom language model. You do not need to pass the weight with each recognition request that uses the custom model.

- For a recognition request, the following example specifies a customization weight of `0.7` with the `POST /v1/recognize` method:

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  --header "Content-Type: audio/flac" \
  --data-binary @audio-file1.flac \
  "{url}/v1/recognize?language_customization_id={customization_id}&customization_weight=0.7"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X POST \
  --header "Authorization: Bearer {token}" \
  --header "Content-Type: audio/flac" \
  --data-binary @audio-file1.flac \
  "{url}/v1/recognize?language_customization_id={customization_id}&customization_weight=0.7"
  ```

  Setting a customization weight during speech recognition overrides a weight that was saved with the model during training.

Troubleshooting the use of custom language models

If you apply a custom language model to speech recognition but find that the service does not appear to be using words that the model contains, check for the following possible problems:

- Make sure that you are correctly passing the customization ID to the recognition request as shown in the previous examples.
- Make sure that the status of the custom model is `available`, meaning that it is fully trained and ready to use. For more information, see Listing custom language models.
- Check the pronunciations that were generated for the new words to make sure that they are correct. For more information, see
  - Validating a words resource for previous-generation models
  - Validating a words resource for large speech models and next-generation models

Working with corpora and custom words for previous-generation models

Note: This information is specific to custom models that are based on *previous-generation models*. For information about corpora and custom words for custom models that are based on *next-generation models*, see Working with corpora and custom words for next-generation models.

You can populate a custom language model with words by adding corpora or grammars to the model, or by adding custom words directly:

- **Corpora** - The recommended means of populating a custom language model with words is to add one or more corpora to the model. When you add a corpus, the service analyzes the file and automatically adds any new words that it finds to the custom model. Adding a corpus to a custom model allows the service to extract domain-specific words in context, which helps ensure better transcription results. For more information, see Working with Corpora.
- **Grammars** - You can add grammars to a custom model to limit speech recognition to the words or phrases that are recognized by a grammar. When you add a grammar to a model, the service automatically adds any new words that it finds to the model, just as it does with corpora. For more information, see Using grammars with custom language models.
- **Individual words** - You can also add individual custom words to a model directly. The service adds the words to the model just as it does words that it discovers from corpora or grammars. When you add a word directly, you can specify multiple pronunciations and indicate how the word is to be displayed. You can also update existing words to modify or augment the definitions that were extracted from corpora or grammars. For more information, see Working with custom words.

Regardless of how you add them, the service stores all words that you add to a custom language model in the model's words resource.

The words resource

The *words resource* includes all words that you add from corpora, from grammars, or directly. Its purpose is to define the pronunciation and spelling of words that are not already present in the service's base vocabulary. The definitions tell the service how to transcribe these *out-of-vocabulary (OOV) words*.

The words resource contains the following information about each OOV word. The service creates the definitions for words that are extracted from corpora and grammars. You specify the characteristics for words that you add or modify directly.

- `word` - The spelling of the word as found in a corpus or grammar or as added by you.

  Important: Do not use characters that need to be URL-encoded. For example, do not use spaces, slashes, backslashes, colons, ampersands, double quotes, plus signs, equals signs, question marks, etc. in the name. The service does not prevent the use of these characters, but because they must be URL-encoded wherever they are used, it is strongly discouraged.

- `sounds_like` - The pronunciation of the word. For words extracted from corpora and grammars, the value represents how the service believes that the word is pronounced based on its language rules. In many cases, the pronunciation reflects the spelling of the `word` field. You can use the `sounds_like` field to modify the word's pronunciation. You can also use the field to specify multiple pronunciations for a word. For more information, see Using the sounds_like field.

- `display_as` - The spelling of the word that the service uses in transcripts. The field indicates how the word is to be displayed. In most cases, the spelling matches the value of the `word` field. You can use the `display_as` field to specify a different spelling for the word. For more information, see Using the display_as field.

- `source` - How the word was added to the words resource. If the service extracted the word from a corpus or grammar, the field lists the name of that resource. Because the service can encounter the same word in multiple resources, the field can list multiple corpus or grammar names. The field includes the string `user` if you add or modify the word directly.

After adding or modifying a word in a model's words resource, it is important that you verify the correctness of the word's definition; for more information, see Validating a words resource for previous-generation models. You must also train the model for the changes to take effect during transcription; for more information, see Train the custom language model.

How much data do I need?

Many factors contribute to the amount of data that you need for an effective custom language model. It is not possible to indicate the exact number of words that you need to add for any custom model or application. Depending on the use case, even adding a few words directly to a custom model can improve the model's quality. But adding OOV words from a corpus that shows the words in the context in which they are used in audio can greatly improve transcription accuracy.

The service limits the number of words that you can add to a custom language model:

- You can add a maximum of 90 thousand OOV words to the words resource of a custom model. This figure includes OOV words from all sources (corpora, grammars, and individual custom words that you add directly).
- You can add a maximum of 10 million total words to a custom model from all sources. This figure includes all words, both OOV words and words that are already part of the service's base vocabulary, that are included in corpora or grammars. For corpora, the service uses these additional words to learn the context in which OOV words can appear, which is why corpora are a more effective means of improving recognition accuracy.

A large words resource can increase the latency of speech recognition, but the exact effect is difficult to quantify or predict. As with the amount of data that is needed to produce an effective custom model, the performance impact of a large words resource depends on many factors. Test your custom model with different amounts of data to determine the performance of your models and data.

Working with corpora for previous-generation models

You use the `POST /v1/customizations/{customization_id}/corpora/{corpus_name}` method to add a corpus to a custom model. A corpus is a plain text file that contains sample sentences from your domain. The following example shows an abbreviated corpus for the healthcare domain. A corpus file is typically much longer.

```
Am I at risk for health problems during travel?
Some people are more likely to have health problems when traveling outside the United States.
How Is Coronary Microvascular Disease Treated?
If you're diagnosed with coronary MVD and also have anemia, you may benefit from treatment for that condition.
Anemia is thought to slow the growth of cells needed to repair damaged blood vessels.
What causes autoimmune hepatitis?
A combination of autoimmunity, environmental triggers, and a genetic predisposition can lead to autoimmune hepatitis.
What research is being done for Spinal Cord Injury?
The National Institute of Neurological Disorders and Stroke NINDS conducts spinal cord research in its laboratories at the
National Institutes of Health NIH.
NINDS also supports additional research through grants to major research institutions across the country.
Some of the more promising rehabilitation techniques are helping spinal cord injury patients become more mobile.
What is Osteogenesis imperfecta OI?
. . .
```

Speech recognition relies on statistical algorithms to analyze audio. Words from a custom model are in competition with words from the service's base vocabulary as well as other words of the model. (Factors such as audio noise and speaker accents also affect the quality of transcription.)

The accuracy of transcription can depend largely on how words are defined in a model and how speakers say them. To improve the service's accuracy, use corpora to provide as many examples as possible of how OOV words are used in the domain. Repeating the OOV words in corpora can improve the quality of the custom language model. How you duplicate the words in corpora depends on how you expect users to say them in the audio that is to be recognized. The more sentences that you add that represent the context in which speakers use words from the domain, the better the service's recognition accuracy.

The service does not apply a simple word-matching algorithm. Its transcription depends on the context in which words are used. When it parses a corpus, the service includes information about n-grams (bi-grams, tri-grams, and so on) from the sentences of the corpus in the custom model. This information helps the service transcribe audio with greater accuracy, and it explains why training a custom model on corpora is more valuable than training it on custom words alone.

For example, accountants adhere to a common set of standards and procedures that are known as Generally Accepted Accounting Principles (GAAP). When you create a custom model for a financial domain, provide sentences that use the term GAAP in context. The sentences help the service distinguish between general phrases such as "the gap between them is small" and domain-centric phrases such as "GAAP provides guidelines for measuring and disclosing financial information."

In general, it is better for corpora to use words in different contexts, which can improve how the service learns the words. However, if users speak the words in only a couple of contexts, then showing the words in other contexts does not improve the quality of the custom model: Speakers never use the words in those contexts. If speakers are likely to use the same phrase frequently, then repeating that phrase in the corpora can improve the quality of the model. (In some cases, even adding a few custom words directly to a custom model can make a positive difference.)

Preparing a corpus text file

Follow these guidelines to prepare a corpus text file:

- Provide a plain text file that is encoded in UTF-8 if it contains non-ASCII characters. The service assumes UTF-8 encoding if it encounters such characters.

  ⚠ **Important:** Make sure that you know the character encoding of your corpus text files. The service preserves the encoding that it finds in the text files. You must use that same encoding when working with custom words in the custom model. For more information, see [Character encoding for custom words](#).

- Use consistent capitalization for words in the corpus. The words resource is case-sensitive. Mix upper- and lowercase letters and use capitalization only when intended.

- Include each sentence of the corpus on its own line, and terminate each line with a carriage return. Including multiple sentences on the same line can degrade accuracy.

- Add personal names as discrete units on separate lines. Do not add the individual elements of a name on separate lines or as individual custom words, and do not include multiple names on the same line of a corpus. The following example shows the correct way to improve recognition accuracy for three names:

```
Gakuto Kutara
Sebastian Leifson
Malcolm Ingersol
```

Include additional contextual information where appropriate, for example, `Doctor Sebastian Leifson` or `President Malcolm Ingersol`. As with all words, duplicating the names multiple times and, if possible, in different contexts can improve recognition accuracy.

- Beware of typographical errors. The service assumes that typographical errors are new words. Unless you correct them before you train the model, the service adds them to the model's vocabulary. Remember the adage *Garbage in, garbage out!*
- More sentences result in better accuracy. But the service does limit a model to a maximum of 10 million total words and 90 thousand OOV words from all sources combined.

The service cannot generate a pronunciation for all words. After adding a corpus, you must validate the words resource to ensure that each OOV word's definition is complete and valid. For more information, see Validating a words resource for previous-generation models.

What happens when I add a corpus file?

When you add a corpus file, the service analyzes the file's contents. It extracts any new (OOV) words that it finds and adds each OOV word to the custom model's words resource. To distill the most meaning from the content, the service tokenizes and parses the data that it reads from a corpus file. The following sections describe how the service parses a corpus file for each supported language.

Parsing of Dutch, English, French, German, Italian, Portuguese, and Spanish

The following descriptions apply to all supported dialects of Dutch, English, French, German, Italian, Portuguese, and Spanish.

- Converts numbers to their equivalent words.

Language	Whole number	Decimal number
Dutch	`500` becomes `vijfhonderd`	`0,15` becomes `nul komma vijftien`
English	`500` becomes `five hundred`	`0.15` becomes `zero point fifteen`
French	`500` becomes `cinq cents`	`0,15` becomes `zéro virgule quinze`
German	`500` becomes `fünfhundert`	`0,15` becomes `null punkt fünfzehn`
Italian	`500` becomes `cinquecento`	`0,15` becomes `zero virgola quindici`
Portuguese	`500` becomes `quinhentos`	`0,15` becomes `zero ponto quinze`
Spanish	`500` becomes `quinientos`	`0,15` becomes `cero coma quince`

Table 1. Examples of number conversion

- Converts tokens that include certain symbols to meaningful string representations. These examples are not exhaustive. The service makes similar adjustments for other characters as needed. (For Spanish, if the dialect is `es-LA`, `$100` and `100$` become `cien pesos`.)

Language	A dollar sign and a number	A euro sign and a number	A percent sign and a number
Dutch	`$100` becomes `honderd dollar`	`€100` becomes `honderd euro`	`100%` becomes `honderd procent`
English	`$100` becomes `one hundred dollars`	`€100` becomes `one hundred euros`	`100%` becomes `one hundred percent`
French	`$100` becomes `cent dollars`	`€100` becomes `cent euros`	`100%` becomes `cent pour cent`
German	`$100` and `100$` become `einhundert dollar`	`€100` and `100€` become `einhundert euro`	`100%` becomes `einhundert prozent`

Italian	`$100` becomes `cento dollari`	`€100` becomes `cento euro`	`100%` becomes `cento per cento`
Portuguese	`$100` and `100$` become `cem dólares`	`€100` and `100€` become `cem euros`	`100%` becomes `cem por cento`
Spanish	`$100` and `100$` become `cien dólares`	`€100` and `100€` become `cien euros`	`100%` becomes `cien por ciento`

Table 2. Examples of symbol conversion

- Processes non-alphanumeric, punctuation, and special characters depending on their context. For example, the service removes a `$` (dollar sign) or `€` (euro symbol) unless it is followed by a number. Processing is context-dependent and consistent across the supported languages.
- Ignores phrases that are enclosed in `( )` (parentheses), `< >` (angle brackets), `[ ]` (square brackets), or `{ }` (curly braces).

Parsing of Japanese

- Converts all characters to full-width characters.
- Converts numbers to their equivalent words, for example, `500` becomes `五百`, and `0.15` becomes `〇・一五`.
- Does not convert tokens that include symbols to equivalent strings, for example, `100%` becomes `百％`.
- Does not automatically remove punctuation. IBM highly recommends that you remove punctuation if your application is transcription-based as opposed to dictation-based.

Parsing of Korean

- Converts numbers to their equivalent words, for example, `10` becomes `십`.
- Removes the following punctuation and special characters: `- ( ) * : . , ' "`. However, not all punctuation and special characters that are removed for other languages are removed for Korean, for example:
  - Removes a period (`.`) symbol only when it occurs at the end of a line of input.
  - Does not remove a tilde (`~`) symbol.
  - Does not remove or otherwise process Unicode wide-character symbols, for example, `…` (triple dot or ellipsis).

  In general, IBM recommends that you remove punctuation, special characters, and Unicode wide-characters before you process a corpus file.
- Does not remove or ignore phrases that are enclosed in `( )` (parentheses), `< >` (angle brackets), `[ ]` (square brackets), or `{ }` (curly braces).
- Converts tokens that include certain symbols to meaningful string representations, for example:
  - `24%` becomes `이십사퍼센트`.
  - `$10` becomes `십달러`.

  This list is not exhaustive. The service makes similar adjustments for other characters as needed.
- For phrases that consist of Latin (English) characters or a mix of Hangul and Latin characters, the service creates OOV words for the phrases exactly as they appear in the corpus file. And it creates sounds-like pronunciations for the words based on Hangul transcriptions.
  - It gives the OOV word `London` a sounds-like of `런던`.
  - It gives the OOV word `IBM홈페이지` a sounds-like of `아이 비 엠 홈페이지`.

Working with custom words for previous-generation models

You can use the `POST /v1/customizations/{customization_id}/words` and `PUT /v1/customizations/{customization_id}/words/{word_name}` methods to add new words to a custom model's words resource. You can also use the methods to modify or augment a word in a words resource.

You might, for instance, need to use the methods to correct a typographical error or other mistake that is made when a word is added from a corpus. You might also need to add sounds-like definitions for an existing word. If you modify an existing word, the new data that you provide overwrites the word's existing definition in the words resource. The rules for adding a word also apply to modifying an existing word.

⚠ **Important:** You are likely to add most custom words from corpora. Make sure that you know the character encoding of your corpus text files. The service preserves the encoding that it finds in the text files. You must use that same encoding when working with custom words in the custom model. For more information, see Character encoding for custom words.

Using the sounds_like field

The `sounds_like` field specifies how a word is pronounced by speakers. By default, the service automatically attempts to complete the field with the word's spelling. But the service cannot generate a pronunciation for all words. After adding or modifying words, you must validate the words resource to ensure that each word's definition is complete and valid. For more information, see [Validating a words resource for previous-generation models]().

You can provide as many as five alternative pronunciations for a word that is difficult to pronounce or that can be pronounced in different ways. Consider using the field to

- *Provide different pronunciations for acronyms.* For example, the acronym `NCAA` can be pronounced as it is spelled or colloquially as *N. C. double A.* The following example adds both of these sounds-like pronunciations for the word `NCAA`:

  IBM Cloud

  ```
  $ curl -X PUT -u "apikey:{apikey}" \
  --header "Content-Type: application/json" \
  --data "{\"sounds_like\": [\"N. C. A. A.\", \"N. C. double A.\"]}" \
  "{url}/v1/customizations/{customization_id}/words/NCAA"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X PUT \
  --header "Authorization: Bearer {token}" \
  --header "Content-Type: application/json" \
  --data "{\"sounds_like\": [\"N. C. A. A.\", \"N. C. double A.\"]}" \
  "{url}/v1/customizations/{customization_id}/words/NCAA"
  ```

- *Handle foreign words.* For example, the French word `garçon` contains a character that is not found in the English language. You can specify a sounds-like of `gaarson`, replacing `ç` with `s`, to tell the service how English speakers would pronounce the word.

The following sections provide language-specific guidelines for specifying a sounds-like pronunciation. Speech recognition uses statistical algorithms to analyze audio, so adding a word does not guarantee that the service transcodes it with complete accuracy. When you add a word, consider how it might be pronounced. Use the `sounds_like` field to provide various pronunciations that reflect how a word can be spoken.

Guidelines for English

Guidelines for Australian, United Kingdom, and United States English:

- Use English alphabetic characters: `a-z` and `A-Z`.
- Use real or made-up words that are pronounceable in English for words that are difficult to pronounce, for example, `shuchesnie` for the word `Sczcesny`.
- Substitute equivalent English letters for non-English letters, for example, `s` for `ç` or `ny` for `ñ`.
- Substitute non-accented letters for accented letters, for example, `a` for `à` or `e` for `è`.
- You can include multiple words that are separated by spaces. The service enforces a maximum of 40 total characters, not including leading or trailing spaces.

Guidelines for Australian and United States English only:

- To pronounce a single letter, use the letter followed by a period. If the period is followed by another character, be sure to use a space between the period and the next character. For example, use `N. C. A. A.`, *not* `N.C.A.A.`
- Use the spelling of numbers, for example, `seventy-five` for `75`.

Guidelines for United Kingdom English only:

- You cannot use periods or dashes in sounds-like pronunciations for UK English.
- To pronounce a single letter, use the letter followed by a space. For example, use `N C A A`, *not* `N. C. A. A.`, `N.C.A.A.`, or `NCAA`.
- Use the spelling of numbers without dashes, for example, `seventy five` for `75`.

Guidelines for Dutch, French, German, Italian, Portuguese, and Spanish

Guidelines for all supported dialects of Dutch, French, German, Italian, Portuguese, and Spanish:

- You cannot use dashes in sounds-like pronunciations.
- Use alphabetic characters that are valid for the language: `a-z` and `A-Z` including valid accented letters.
- To pronounce a single letter, use the letter followed by a period. If the period is followed by another character, be sure to use a space between the period and the next character. For example, use `N. C. A. A.`, *not* `N.C.A.A.`

- Use real or made-up words that are pronounceable in the language for words that are difficult to pronounce.
- Use the spelling of numbers without dashes, for example, for `75` use
  - *Dutch (Netherlands):* `vijfenzeventig`
  - *French:* `soixante quinze`
  - *German:* `fünfundsiebzig`
  - *Italian:* `settantacinque`
  - *Portuguese (Brazilian):* `setenta e cinco`
  - *Spanish:* `setenta y cinco`
- You can include multiple words that are separated by spaces. The service enforces a maximum of 40 total characters, not including leading or trailing spaces.

Guidelines for Japanese

- Use only full-width Katakana characters by using the `ー` lengthen symbol (*chou-on*, or 長音, in Japanese). Do not use half-width characters.

- Use contracted sounds (*yoh-on*, or 拗音, in Japanese) only in the following syllable contexts:

  `イェ`, `ウィ`, `ウェ`, `ウォ`, `キィ`, `キャ`, `キュ`, `キョ`, `ギャ`, `ギュ`, `ギョ`, `クァ`, `クィ`, `クェ`, `クォ`

  `グァ`, `グォ`, `シィ`, `シェ`, `シャ`, `シュ`, `ショ`, `ジィ`, `ジェ`, `ジャ`, `ジュ`, `ジョ`, `スィ`, `ズィ`, `チェ`

  `チャ`, `チュ`, `チョ`, `ヂェ`, `ヂャ`, `ヂュ`, `ヂョ`, `ツァ`, `ツィ`, `ツェ`, `ツォ`, `ティ`, `テュ`, `ディ`, `デャ`

  `デュ`, `デョ`, `トゥ`, `ドゥ`, `ニェ`, `ニャ`, `ニュ`, `ニョ`, `ヒャ`, `ヒュ`, `ヒョ`, `ビャ`, `ビュ`, `ビョ`, `ピィ`

  `ピャ`, `ピュ`, `ピョ`, `ファ`, `フィ`, `フェ`, `フォ`, `フュ`, `ミャ`, `ミュ`, `ミョ`, `リィ`, `リェ`, `リャ`, `リュ`

  `リョ`, `ヴァ`, `ヴィ`, `ヴェ`, `ヴォ`, `ヴュ`

- Use only the following syllables after an assimilated sound (*soku-on*, or 促音, in Japanese):

  `バ`, `ビ`, `ブ`, `ベ`, `ボ`, `チ`, `チェ`, `チャ`, `チュ`, `チョ`, `ダ`, `デ`, `ディ`, `ド`, `ドゥ`, `フ`

  `ファ`, `フィ`, `フェ`, `フォ`, `ガ`, `ギ`, `グ`, `ゲ`, `ゴ`, `ハ`, `ヒ`, `ヘ`, `ホ`, `ジ`, `ジェ`, `ジャ`

  `ジュ`, `ジョ`, `カ`, `キ`, `ク`, `ケ`, `コ`, `キャ`, `キュ`, `キョ`, `パ`, `ピ`, `プ`, `ペ`, `ポ`, `ピャ`

  `ピュ`, `ピョ`, `サ`, `ス`, `セ`, `ソ`, `シ`, `シェ`, `シャ`, `シュ`, `ショ`, `タ`, `テ`, `ト`, `ツ`, `ザ`

  `ズ`, `ゼ`, `ゾ`

- Do not use `ン` as the first character of a word. For example, use `ウーント` instead of `ンート`, the latter of which is invalid.

- Many compound words consist of *prefix+noun* or *noun+suffix*. The service's base vocabulary covers most compound words that occur frequently (for example, `長電話` and `古新聞`) but not those compound words that occur infrequently. If your corpus commonly contains compound words, add them as one word as the first step of your customization. For example, `古鉛筆` is not common in general Japanese text; if you use it often, add it to your custom model to improve transcription accuracy.

- Do not use a trailing assimilated sound.

Guidelines for Korean

- Use Korean Hangul characters, symbols, and syllables.
- You can also use Latin (English) alphabetic characters: `a-z` and `A-Z`.
- Do not use any characters or symbols that are not included in the previous sets.

Using the display_as field

The `display_as` field specifies how a word is displayed in a transcript. It is intended for cases where you want the service to display a string that is different from the word's spelling. For example, you can indicate that the word `hhonors` is to be displayed as `HHonors` regardless of whether it sounds like `hilton honors` or `h honors`.

IBM Cloud

```
$ curl -X PUT -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--data "{\"sounds_like\": [\"hilton honors\", \"H. honors\"], \"display_as\": \"HHonors\"}" \
"{url}/v1/customizations/{customization_id}/words/hhonors"
```

IBM Cloud Pak for Data

```
$ curl -X PUT \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--data "{\"sounds_like\": [\"hilton honors\", \"H. honors\"], \"display_as\": \"HHonors\"}" \
"{url}/v1/customizations/{customization_id}/words/hhonors"
```

As another example, you can indicate that the word `IBM` is to be displayed as `IBM™`.

IBM Cloud

```
$ curl -X PUT -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--data "{\"sounds_like\": [\"I. B. M.\"], \"display_as\":\"IBM™\"}" \
"{url}/v1/customizations/{customization_id}/words/IBM"
```

IBM Cloud Pak for Data

```
$ curl -X PUT \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--data "{\"sounds_like\": [\"I. B. M.\"], \"display_as\":\"IBM™\"}" \
"{url}/v1/customizations/{customization_id}/words/IBM"
```

Interaction with smart formatting and numeric redaction

If you use the `smart_formatting` or `redaction` parameters with a recognition request, be aware that the service applies smart formatting and redaction to a word before it considers the `display_as` field for the word. You might need to experiment with results to ensure that the features do not interfere with how your custom words are displayed. You might also need to add custom words to accommodate the effects.

For instance, suppose that you add the custom word `one` with a `display_as` field of `one`. Smart formatting changes the word `one` to the number `1`, and the display-as value is not applied. To work around this issue, you could add a custom word for the number `1` and apply the same `display_as` field to that word.

For more information about working with these features, see Smart formatting and Numeric redaction.

What happens when I add or modify a custom word?

How the service responds to a request to add or modify a custom word depends on the fields and values that you specify. It also depends on whether the word exists in the service's base vocabulary.

- **Omit both the `sounds_like` and `display_as` fields:**
  - *If the word does not exist in the service's base vocabulary,* the service attempts to set the `sounds_like` field to its pronunciation of the word. It cannot generate a pronunciation for all words, so you must review the word's definition to ensure that it is complete and valid. The service sets the `display_as` field to the value of the `word` field.
  - *If the word exists in the service's base vocabulary,* the service leaves the `sounds_like` and `display_as` fields empty. These fields are empty only if the word exists in the service's base vocabulary. The word's presence in the model's words resource is harmless but unnecessary.
- **Specify only the `sounds_like` field:**
  - *If the `sounds_like` field is valid,* the service sets the `display_as` field to the value of the `word` field.
  - *If the `sounds_like` field is invalid:*
    - The `POST /v1/customizations/{customization_id}/words` method adds an `error` field to the word in the model's words resource.
    - The `PUT /v1/customizations/{customization_id}/words/{word_name}` method fails with a 400 response code and an error message. The service does not add the word to the words resource.
- **Specify only the `display_as` field:**
  - *If the word does not exist in the service's base vocabulary,* the service attempts to set the `sounds_like` field to its pronunciation of the word. It cannot generate a pronunciation for all words, so you must review the word's definition to ensure that it is complete and valid. The service leaves the `display_as` field as specified.

    - *If the word exists in the service's base vocabulary,* the service leaves the `sounds_like` empty and leaves the `display_as` field as specified.
- **Specify both the `sounds_like` and `display_as` fields:**
    - *If the `sounds_like` field is valid,* the service sets the `sounds_like` and `display_as` fields to the specified values.
    - *If the `sounds_like` field is invalid,* the service responds as it does in the case where the `sounds_like` field is specified but the `display_as` field is not.

Validating a words resource for previous-generation models

Especially when you add a corpus to a custom language model or add multiple custom words at one time, examine the OOV words in the model's words resource.

- *Look for typographical and other errors.* Especially when you add corpora, which can be large, mistakes are easily made. Typographical errors in a corpus fole (or in custom words or a grammar file) have the unintended consequence of adding new words to a model's words resource, as do ill-formed HTML tags that are left in a corpus file.
- *Verify the sounds-like pronunciations.* The service tries to generate sounds-like pronunciations for OOV words automatically. In most cases, these pronunciations are sufficient. But the service cannot generate a pronunciation for all words, so you must review the word's definition to ensure that it is complete and valid. Reviewing the pronunciations for accuracy is also recommended for words that have unusual spellings or are difficult to pronounce, and for acronyms and technical terms.

To validate and, if necessary, correct a word for a custom model, regardless of how it was added to the words resource, use the following methods:

- List all of the words from a custom model by using the `GET /v1/customizations/{customization_id}/words` method or query an individual word with the `GET /v1/customizations/{customization_id}/words/{word_name}` method. For more information, see Listing custom words from a custom language model.
- Modify words in a custom model to correct errors or to add sounds-like or display-as values by using the `POST /v1/customizations/{customization_id}/words` or `PUT /v1/customizations/{customization_id}/words/{word_name}` method. For more information, see Working with custom words for previous-generation models.
- Delete extraneous words that are introduced in error (for example, by typographical or other mistakes in a corpus) by using the `DELETE /v1/customizations/{customization_id}/words/{word_name}` method. For more information, see Deleting a word from a custom language model.
    - If the word was extracted from a corpus, you can instead update the corpus text file to correct the error and then reload the file by using the `allow_overwrite` parameter of the `POST /v1/customizations/{customization_id}/corpora/{corpus_name}` method. For more information, see Add a corpus to the custom language model.
    - If the word was extracted from a grammar, you can update the grammar file to correct the error and then reload the file by using the `allow_overwrite` parameter of the `POST /v1/customizations/{customization_id}/grammars/{grammar_name}` method. For more information, see Add a grammar to the custom language model.

Working with corpora and custom words for large speech models and next-generation models

Note: This information is specific to custom models that are based on *large speech models and next-generation models*. For information about corpora and custom words for custom models that are based on *previous-generation models*, see Working with corpora and custom words for previous-generation models.

You populate a custom language model with words by adding corpora to the model or by adding custom words directly to the model. You use the same methods and operations for large speech models, previous- and next-generation models. For more information about adding corpora and custom words to a model, see Working with corpora for large speech models and next-generation models and Working with custom words for large speech models and next-generation models.

Although language model customization is similar in usage and intent for large speech models, previous- and next-generation models, there are differences between the three types of models at the implementation level. To understand how language model customization works for large speech models and next-generation models, and how you can make the best use of customization, you need a high-level understanding of the differences.

- When you create and use a custom language model that is based on a *large speech model or previous-generation model*, the service relies on words from the custom model to create transcripts that contain domain-specific terms. In combination with words from its base vocabulary, the service uses these words from the custom model to predict and transcribe speech from audio. You provide the information for a custom language model in the form of corpora, custom words, and grammars. The service stores this information in the words resource for the custom model.

- When you create a custom language model that is based on a *next-generation model*, the services relies on sequences of characters from the custom model to create transcripts that reflect domain-specific terms. In combination with character sequences from the base model, the service uses these series of characters from the custom model to predict and transcribe speech from audio.

  You provide the information for a custom language model in the form of corpora, custom words, and grammars. But rather than rely on a words resource that contains these words, the service extracts and stores character sequences from the corpora and custom words. The service does not extract and calculate out-of-vocabulary (OOV) words from corpora and custom words. The words resource is simply where it stores custom words that you add directly to the model.

When you develop a custom language model based on a large speech model or next-generation model, you must still provide corpora and custom words to train the model on domain-specific terminology. So the process of creating and training a custom model is largely the same for large speech models, next-generation and previous-generation models.

The following topics describe the rules for providing corpora and custom words for a custom language model that is based on large speech models and next-generation models. The rules are similar to those for working with a custom model based on a previous-generation model, but some important differences do exist.

The words resource

The *words resource* includes custom words that you add directly to the custom model. The words resource contains the following information about each custom word:

- `word` - The spelling of the word as added by you.

  Important: Do not use characters that need to be URL-encoded. For example, do not use spaces, slashes, backslashes, colons, ampersands, double quotes, plus signs, equals signs, question marks, etc. in the name. The service does not prevent the use of these characters, but because they must be URL-encoded wherever used, it is strongly discouraged.

- `sounds_like` - The pronunciation of the word. You can use the `sounds_like` field to add one or more pronunciations for the word. For more information, see Using the sounds_like field.
- `display_as` - The spelling of the word that the service uses in transcripts. Unless you specify an alternative representation, the spelling matches the value of the `word` field. For more information, see Using the display_as field.
- `source` - How the word was added to the words resource. The field always contains the string `user` to indicate that it was added directly as a custom word.

After adding or modifying a custom word, it is important that you verify the correctness of the word's definition; for more information, see Validating a words resource for large speech models and next-generation models. You must also train the model for the changes to take effect during transcription; for more information, see Train the custom language model.

Note: You can add custom words that already exist, for example, to add more sounds-like pronunciations for common words. Otherwise, there is no reason to duplicate common words. Such words remain in the model's words resource, but they are harmless and unnecessary.

- `mapping_only` - Parameter for custom words. You can use the 'mapping_only' key in custom words as a form of post processing. This key parameter has a boolean value to determine whether 'sounds_like' (for non-Japanese models) or word (for Japanese) is not used for the model fine-tuning, but for the replacement for 'display_as'. This feature helps you when you use custom words exclusively to map 'sounds_like' (or word) to 'display_as' value. When you use custom words solely for post-processing purposes that does not need fine-tuning.

Use case examples,

Before using 'mapping_only': Speech to Text machine output is 'hilton honors' as its ASR transcript. However, you want it to be displayed as 'HHonors' as final output. So, you can use the following custom word to map 'hilton honors' to 'HHonors'.

```
{"word": "HHonors", "sounds_like": ["hilton honors"], "display_as": "HHonors"}
```

While this maps any word 'hilton honors' in ASR transcript to 'HHonors', it fine-tunes the model with 'sounds_like' (hilton honors) by default, even if the model has no problem with recognizing the word 'hilton honors'. This is the example of words that do not need to be fine-tuned but need to be mapped to 'display_as'.

After using 'mapping_only': Since Speech to Text model is recognizing the word 'hilton honors' very well, it does not have to be fine-tuned on that word. Thus, you can use the following custom words to skip training and mapping the 'sounds_like' to 'display_as'.

```
{"word": "HHonors", "sounds_like": ["hilton honors"], "display_as": "HHonors", "mapping_only": true}
```

This parameter is applicable for the next-generation models that support the enhanced customization (English models, ja-Jp models and so on). See the list of supported models.

How much data do I need?

Many factors contribute to the amount of data that you need for an effective custom language model. It is not possible to indicate the exact amount of data that you need to add for any custom model or application. Depending on the use case, even adding a few words directly to a custom model can improve the model's quality. But adding corpora that show the words in the context in which they are used can greatly improve transcription accuracy.

You can add a maximum of 10 million total words to a custom model from all sources. This figure includes all words that are included in corpora and that you add directly. The service uses all of the words from a corpus to learn the context in which sequences of characters can occur, which is why corpora are a more effective means of improving recognition accuracy.

Adding a large number of corpora and words can increase the latency of speech recognition, but the exact effect is difficult to quantify or predict. As with the amount of data that is needed to produce an effective custom model, the performance impact of a large amount of data depends on many factors. Test your custom model with different amounts of data to determine the performance of your models.

Guidelines for adding words to custom models based on improved next-generation models

For custom models that are based on next-generation language models that use improved customization, limit the number of custom words that you add directly to the model with the following methods:

- `POST /v1/customizations/{customization_id}/words`
- `PUT /v1/customizations/{customization_id}/words/{word_name}`

Using these methods to add custom words to a custom model can significantly increase the training time of the model. Training time increases linearly with the total number of words that you add. However, the training time increases only when the model is first trained on the new custom words. The time required for subsequent training requests, without new custom words, returns to normal.

Training a custom model with words added only via corpora with the following method is typically quick:

- `POST /v1/customizations/{customization_id}/corpora/{corpus_name}`

For more information about improved customization for next-generation models, see Improved language model customization for next-generation models.

Note: The non-Japanese models use 'sounds_like' for mapping ('sounds_like' -> 'display_as').

Guidelines for adding words to Japanese models based on improved next-generation models

Do not add custom words for known words, words that are commonly recognized and have a general correspondence between the word and how it is pronounced. Use custom words to create a mapping between how less common words are spelled and how they are pronounced. Also use custom words to add unknown words, those that lack a correspondence between the word and its pronunciation or that are not commonly recognized as words. Adding such words to a corpus is equally effective.

The service enforces a maximum limit of 25 total characters, not including leading or trailing spaces, for custom words and sounds-likes. If you add a custom word or sounds-like that exceeds this limit, the service automatically treats the word as if it were added by a corpus. The word does not appear as a custom word for the model. For the most effective training, it is recommended that Japanese custom words and sounds-likes contain no more than 20 characters. Add long words such as `ＩＢＭクラウド音声認識サービス` to a corpus.

For example, the pronunciation `アイビーエム` for the word `ＩＢＭ` is recognized without customization, so there is no need to add it as a custom word. Because `ＩＢＭ`, `クラウド`, `音声認識`, and `サービス` are common words, adding them as custom words has no effect.

For a custom word, enter a commonly used notation that reflects the word's usage and pronunciation. This allows for more efficient customization. For instance, the following example does not reliably produce the string `Artificial_Intelligence` in response to the utterance `エーアイ` because `Artificial_Intelligence` and `人工知能` are not generally uttered as `AI`:

```
{\"word\": \"Artificial_Intelligence\", \"sounds_like\": [\"エーアイ\"], \"display_as\": \"Artificial_Intelligence\"},
{\"word\": \"人工知能\", \"sounds_like\": [\"エーアイ\"], \"display_as\": \"Artificial_Intelligence\"}
```

Because the context is typically something like `これからＡＩはますます発展してきます`, `ＡＩ` is the most appropriate notation for the sounds-like `エーアイ`. The following example is therefore likely to yield better results:

```
{\"word\": \"ＡＩ\", \"sounds_like\": [\"エーアイ\"], \"display_as\": \"Artificial_Intelligence\"}
```

Finally, in custom words, half-width alphabetic characters are converted to full-width characters. English uppercase and lowercase characters are treated as different characters.

Note: Japanese models use word for mapping ('word' -> 'display_as').

Working with corpora for large speech models and next-generation models

You use the `POST /v1/customizations/{customization_id}/corpora/{corpus_name}` method to add a corpus to a custom model. A corpus is a plain text file that contains sample sentences from your domain. The following example shows an abbreviated corpus for the healthcare domain. A corpus file is typically much longer.

```
Am I at risk for health problems during travel?
Some people are more likely to have health problems when traveling outside the United States.
How Is Coronary Microvascular Disease Treated?
If you're diagnosed with coronary MVD and also have anemia, you may benefit from treatment for that condition.
Anemia is thought to slow the growth of cells needed to repair damaged blood vessels.
What causes autoimmune hepatitis?
A combination of autoimmunity, environmental triggers, and a genetic predisposition can lead to autoimmune hepatitis.
What research is being done for Spinal Cord Injury?
The National Institute of Neurological Disorders and Stroke NINDS conducts spinal cord research in its laboratories at the
National Institutes of Health NIH.
NINDS also supports additional research through grants to major research institutions across the country.
Some of the more promising rehabilitation techniques are helping spinal cord injury patients become more mobile.
What is Osteogenesis imperfecta OI?
. . .
```

Character sequences in words from a custom model are in competition with character sequences from the base model, as well as sequences from other words of the model. (Factors such as audio noise and speaker accents also affect the quality of transcription.)

The accuracy of transcription can depend largely on the data that you add to a model and how speakers say words in audio. To improve the service's accuracy, use corpora to provide as many examples as possible of how words are used in a domain. Repeating the words in corpora can improve the quality of a custom language model. How you duplicate the words in corpora depends on how you expect users to say them in the audio that is to be recognized. The more sentences that you add that represent the context in which speakers use words from the domain, the better the service's recognition accuracy.

For example, accountants adhere to a common set of standards and procedures that are known as Generally Accepted Accounting Principles (GAAP). When you create a custom model for a financial domain, provide sentences that use the term GAAP in context. The sentences help the service distinguish between general phrases such as "the gap between them is small" and domain-centric phrases such as "GAAP provides guidelines for measuring and disclosing financial information."

In general, it is better for corpora to use words in different contexts, which can improve how the service learns the phrases. However, if users speak the words in only a couple of contexts, then showing the words in other contexts does not improve the quality of the custom model: Speakers never use the words in those contexts. If speakers are likely to use the same phrase frequently, repeating that phrase in the corpora can improve the quality of the model. In some cases, even adding a few custom words directly to a custom model can make a positive difference.

Preparing a corpus text file

Follow these guidelines to prepare a corpus text file:

- Provide a plain text file that is encoded in UTF-8 if it contains non-ASCII characters. The service assumes UTF-8 encoding if it encounters such characters.

  ⚠ **Important:** Make sure that you know the character encoding of your corpus text files. The service preserves the encoding that it finds in the text files. You must use that same encoding when working with custom words in the custom model. For more information, see Character encoding for custom words.

- Use consistent capitalization for words in the corpus. Mix upper- and lowercase letters and use capitalization only when intended.

- Include each sentence of the corpus on its own line, and terminate each line with a carriage return. Including multiple sentences on the same line can degrade accuracy.

- Add personal names as discrete units on separate lines. Do not add the individual elements of a name on separate lines or as individual custom words, and do not include multiple names on the same line of a corpus. The following example shows the correct way to improve recognition accuracy for three names:

  ```
  Gakuto Kutara
  Sebastian Leifson
  Malcolm Ingersol
  ```

  Include additional contextual information where appropriate, for example, `Doctor Sebastian Leifson` or `President Malcolm Ingersol`. As with all words, duplicating the names multiple times can improve recognition accuracy.

- Beware of typographical errors. The service assumes that typographical errors are new words. Remember the adage *Garbage in, garbage out!*
- More sentences result in better accuracy. But the service does limit a model to a maximum of 10 million total words from all sources combined.

What happens when I add a corpus file?

When you add a corpus file, the service analyzes the file's contents. To distill the most meaning from the content, the service tokenizes and parses the data that it reads from a corpus file. The following topics describe how the service parses a corpus file for each supported language.

Note: Information for the following languages is not yet available: Arabic, Chinese, Czech, Hindi, and Swedish. If you need this information for your custom language model, contact your IBM Support representative.

Parsing of Dutch, English, French, German, Italian, Portuguese, and Spanish

The following information applies to all supported dialects of Dutch, English, French, German, Italian, Portuguese, and Spanish:

- Converts numbers to their equivalent words.

Language	Whole number	Decimal number
Dutch	`500` becomes `vijfhonderd`	`0,15` becomes `nul komma vijftien`
English	`500` becomes `five hundred`	`0.15` becomes `zero point fifteen`
French	`500` becomes `cinq cents`	`0,15` becomes `zéro virgule quinze`
German	`500` becomes `fünfhundert`	`0,15` becomes `null punkt fünfzehn`
Italian	`500` becomes `cinquecento`	`0,15` becomes `zero virgola quindici`
Portuguese	`500` becomes `quinhentos`	`0,15` becomes `zero ponto quinze`
Spanish	`500` becomes `quinientos`	`0,15` becomes `cero coma quince`

Table 1. Examples of number conversion

- Converts tokens that include certain symbols to meaningful string representations. These examples are not exhaustive. The service makes similar adjustments for other characters as needed.

Language	A dollar sign and a number	A euro sign and a number	A percent sign and a number
Dutch	`$100` becomes `honderd dollar`	`€100` becomes `honderd euro`	`100%` becomes `honderd procent`
English	`$100` becomes `one hundred dollars`	`€100` becomes `one hundred euros`	`100%` becomes `one hundred percent`
French	`$100` becomes `cent dollars`	`€100` becomes `cent euros`	`100%` becomes `cent pour cent`
German	`$100` and `100$` become `einhundert dollar`	`€100` and `100€` become `einhundert euro`	`100%` becomes `einhundert prozent`
Italian	`$100` becomes `cento dollari`	`€100` becomes `cento euro`	`100%` becomes `cento per cento`
Portuguese	`$100` and `100$` become `cem dólares`	`€100` and `100€` become `cem euros`	`100%` becomes `cem por cento`
Spanish	`$100` and `100$` become `cien dólares`	`€100` and `100€` become `cien euros`	`100%` becomes `cien por ciento`

Table 2. Examples of symbol conversion

- Processes non-alphanumeric, punctuation, and special characters depending on their context. For example, the service removes a `$` (dollar sign) or `€` (euro symbol) unless it is followed by a number. Processing is context-dependent and consistent across the supported languages.

- Ignores phrases that are enclosed in `( )` (parentheses), `< >` (angle brackets), `[ ]` (square brackets), or `{ }` (curly braces).

Parsing of Japanese

The following information applies to Japanese:

- Converts all characters to full-width characters.
- Converts numbers to their equivalent words, for example, `500` becomes `五百`, and `0.15` becomes `〇・一五`.
- Does not convert tokens that include symbols to equivalent strings, for example, `100%` becomes `百%`.
- Does not automatically remove punctuation. IBM highly recommends that you remove punctuation if your application is transcription-based as opposed to dictation-based.

Parsing of Korean

The following information applies to Korean:

- Converts numbers to their equivalent words, for example, `10` becomes `십`.
- Removes the following punctuation and special characters: `- ( ) * : . , ' "`. However, not all punctuation and special characters that are removed for other languages are removed for Korean, for example:
    - Removes a period (`.`) symbol only when it occurs at the end of a line of input.
    - Does not remove a tilde (`~`) symbol.
    - Does not remove or otherwise process Unicode wide-character symbols, for example, `…` (triple dot or ellipsis).

  In general, IBM recommends that you remove punctuation, special characters, and Unicode wide-characters before you process a corpus file.
- Does not remove or ignore phrases that are enclosed in `( )` (parentheses), `< >` (angle brackets), `[ ]` (square brackets), or `{ }` (curly braces).
- Converts tokens that include certain symbols to meaningful string representations, for example:
    - `24%` becomes `이십사퍼센트`.
    - `$10` becomes `십달러`.

  This list is not exhaustive. The service makes similar adjustments for other characters as needed.
- For phrases that consist of Latin (English) characters or a mix of Hangul and Latin characters, the service uses the phrases exactly as they appear in the corpus file.

Working with custom words for large speech models and next-generation models

You can use the `POST /v1/customizations/{customization_id}/words` and `PUT /v1/customizations/{customization_id}/words/{word_name}` methods to add new words to a custom model. You can also use the methods to modify or augment a custom word.

You might, for instance, need to use the methods to correct a typographical error or other mistake that is made when a word is added to a custom model. If you modify an existing word, the new data that you provide overwrites the word's existing definition in the words resource. The rules for adding a word also apply to modifying an existing word.

Important: You are likely to add most custom words from corpora. Make sure that you know the character encoding of your corpus text files. The service preserves the encoding that it finds in the text files. You must use that same encoding when working with custom words in the custom model. For more information, see Character encoding for custom words.

Using the sounds_like field

The `sounds_like` field specifies how a word is pronounced by speakers in audio. By default, the service does not automatically attempt to generate a sounds-like pronunciation for a word for which you do not provide one. You can add sounds-likes for any words that do not have them. After adding or modifying words, you must validate the words resource to ensure that each word's definition is complete and valid. For more information, see Validating a words resource for large speech models and next-generation models.

You can provide as many as five alternative pronunciations for a word that is difficult to pronounce or that can be pronounced in different ways. Some possible uses of the field follow:

- *Provide different pronunciations for acronyms.* For example, the acronym `NCAA` can be pronounced as it is spelled or colloquially as *N C double A* The following example adds both of these sounds-like pronunciations for the word `NCAA`:

  IBM Cloud

```
$ curl -X PUT -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--data "{\"sounds_like\": [\"N C A A\", \"N C double A\"]}" \
"{url}/v1/customizations/{customization_id}/words/NCAA"
```

IBM Cloud Pak for Data

```
$ curl -X PUT \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--data "{\"sounds_like\": [\"N C A A\", \"N C double A\"]}" \
"{url}/v1/customizations/{customization_id}/words/NCAA"
```

For more information about how the service recognizes acronyms, see Additional transcription efforts.

- *Handle foreign words.* For example, the French word `garçon` contains a character that is not found in the English language. You can specify a sounds-like of `gaarson`, replacing `ç` with `s`, to tell the service how English speakers would pronounce the word.

The following topics provide guidelines for specifying a sounds-like pronunciation. Speech recognition uses statistical algorithms to analyze audio, so adding a word does not guarantee that the service transcodes it with complete accuracy. When you add a word, consider how it might be pronounced. Use the `sounds_like` field to provide various pronunciations that reflect how a word can be spoken.

Note: Information for the following languages is not yet available: Arabic, Chinese, Czech, Hindi, and Swedish. If you need this information for your custom language model, contact your IBM Support representative.

Note: The non-Japanese models use 'sounds_like' for mapping ('sounds_like' -> 'display_as'). Japanese models use word for mapping ('word' -> 'display_as').

General guidelines for all languages

Follow these guidelines when specifying a sounds-like for any language:

- *Do not use punctuation characters in sounds-likes.* For example, do not use periods, dashes, underscores, commas, end of sentence punctuation characters, and special characters such as dollar and euro signs, parentheses, brackets, and braces.
- *Use alphabetic characters that are valid for your language.* For English, this includes `a-z` and `A-Z`. For other languages, valid characters can include accented letters or language-specific characters.
- *In English, substitute equivalent English letters for non-English or accented letters.* For example, `s` for `ç`, `ny` for `ñ`, or `e` for `è`.
- *Use real or made-up words that are pronounceable for words that are difficult to pronounce.* For example, in English you can use the sounds-like `shuchesnie` for the word `Sczcesny`.
- *Use the spelling of numbers without dashes.* For example, for the number `75`, use `seventy five` in English, `setenta y cinco` in Spanish, and `soixante quinze` in French.
- *To pronounce a single letter, use the letter followed by a space.* For example, use `N C A A`, *not* `N. C. A. A.`, `N.C.A.A.`, or `NCAA`.
- *You can include multiple words that are separated by spaces.*
  - For most languages, the service enforces a maximum of 40 total characters, not including leading or trailing spaces.
  - For Japanese, the service enforces a maximum limit of 25 total characters, not including leading or trailing spaces. If you add a custom word or sounds-like that exceeds this limit, the service automatically treats the word as if it were added by a corpus. The word does not appear as a custom word for the model. For the most effective training, it is recommended that Japanese custom words and sounds-likes contain no more than 20 characters.

Guidelines for Japanese

Follow these guidelines when specifying a sounds-like for Japanese:

- Use only full-width Katakana characters by using the `ー` lengthen symbol (*chou-on*, or 長音, in Japanese). Do not use half-width characters. (For the `display_as` field, if you enter a half-width character, it is rendered as a half-width character in transcription results.)

- Use contracted sounds (*yoh-on*, or 拗音, in Japanese) only in the following syllable contexts:

  `イェ`, `ウィ`, `ウェ`, `ウォ`, `キィ`, `キャ`, `キュ`, `キョ`, `ギャ`, `ギュ`, `ギョ`, `クァ`, `クィ`, `クェ`, `クォ`

  `グァ`, `グォ`, `シィ`, `シェ`, `シャ`, `シュ`, `ショ`, `ジィ`, `ジェ`, `ジャ`, `ジュ`, `ジョ`, `スィ`, `ズィ`, `チェ`

`チャ`, `チュ`, `チョ`, `ヂェ`, `ヂャ`, `ヂュ`, `ヂョ`, `ツァ`, `ツィ`, `ツェ`, `ツォ`, `ティ`, `テュ`, `ディ`, `デャ`

`デュ`, `デョ`, `トゥ`, `ドゥ`, `ニェ`, `ニャ`, `ニュ`, `ニョ`, `ヒャ`, `ヒュ`, `ヒョ`, `ビャ`, `ビュ`, `ビョ`, `ピィ`

`ピャ`, `ピュ`, `ピョ`, `ファ`, `フィ`, `フェ`, `フォ`, `フュ`, `ミャ`, `ミュ`, `ミョ`, `リィ`, `リェ`, `リャ`, `リュ`

`リョ`, `ヴァ`, `ヴィ`, `ヴェ`, `ヴォ`, `ヴュ`

- Use only the following syllables after an assimilated sound (*soku-on*, or 促音, in Japanese):

  `バ`, `ビ`, `ブ`, `ベ`, `ボ`, `チ`, `チェ`, `チャ`, `チュ`, `チョ`, `ダ`, `デ`, `ディ`, `ド`, `ドゥ`, `フ`

  `ファ`, `フィ`, `フェ`, `フォ`, `ガ`, `ギ`, `グ`, `ゲ`, `ゴ`, `ハ`, `ヒ`, `ヘ`, `ホ`, `ジ`, `ジェ`, `ジャ`

  `ジュ`, `ジョ`, `カ`, `キ`, `ク`, `ケ`, `コ`, `キャ`, `キュ`, `キョ`, `パ`, `ピ`, `プ`, `ペ`, `ポ`, `ピャ`

  `ピュ`, `ピョ`, `サ`, `ス`, `セ`, `ソ`, `シ`, `シェ`, `シャ`, `シュ`, `ショ`, `タ`, `テ`, `ト`, `ツ`, `ザ`

  `ズ`, `ゼ`, `ゾ`

- Do not use `ン` as the first character of a word. For example, use `ウーント` instead of `ソート`, the latter of which is invalid.
- The character-sequence `ウー` is ambiguous in some left contexts. Do not use characters (syllables) that end with the phoneme `/o/`, such as `ロ` and `ト`. In such cases, use `ウウ` or just `ウ` instead of `ウー`. For example, use `ロウウマン` or `ロウマン` instead of `ロウーマン`.
- Many compound words consist of *prefix+noun* or *noun+suffix*. The character sequences of the base model cover most compound words that occur frequently (for example, `長電話` and `古新聞`) but not those compound words that occur infrequently. If your corpus commonly contains compound words, add them as one word as the first step of your customization. For example, `古鉛筆` is not common in general Japanese text; if you use it often, add it to your custom model to improve transcription accuracy.
- Do not use a trailing assimilated sound.

Guidelines for Korean

Follow these guidelines when specifying a sounds-like for Korean:

- Use Korean Hangul characters, symbols, and syllables.
- You can also use Latin (English) alphabetic characters: `a-z` and `A-Z`.
- Do not use any characters or symbols that are not included in the previous sets.

Using the display_as field

The `display_as` field specifies how a word is displayed in a transcript. By default, the service sets the field to match the spelling of the custom word. The field is intended for cases where you want the service to display a string that is different from the word's spelling. For example, you can indicate that the word `hhonors` is to be displayed as `HHonors`.

IBM Cloud

```
$ curl -X PUT -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--data "{\"display_as\": \"HHonors\"}" \
"{url}/v1/customizations/{customization_id}/words/hhonors"
```

IBM Cloud Pak for Data

```
$ curl -X PUT \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--data "{\"display_as\": \"HHonors\"}" \
"{url}/v1/customizations/{customization_id}/words/hhonors"
```

As another example, you can indicate that the word `IBM` is to be displayed as `IBM™`.

IBM Cloud

```
$ curl -X PUT -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--data "{\"display_as\":\"IBM™\"}" \
"{url}/v1/customizations/{customization_id}/words/IBM"
```

IBM Cloud Pak for Data

```
$ curl -X PUT \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--data "{\"display_as\":\"IBM™\"}" \
"{url}/v1/customizations/{customization_id}/words/IBM"
```

Note: The non-Japanese models use 'sounds_like' for mapping ('sounds_like' -> 'display_as'). Japanese models use word for mapping ('word' -> 'display_as').

Interaction with smart formatting and numeric redaction

If you use the `smart_formatting` or `redaction` parameters with a recognition request, be aware that the service applies smart formatting and redaction to a word before it considers the `display_as` field for the word. You might need to experiment with results to ensure that the features do not interfere with how your custom words are displayed. You might also need to add custom words to accommodate the effects.

For instance, suppose that you add the custom word `one` with a `display_as` field of `one`. Smart formatting changes the word `one` to the number `1`, and the display-as value is not applied. To work around this issue, you could add a custom word for the number `1` and apply the same `display_as` field to that word.

For more information about working with these features, see Smart formatting and Numeric redaction.

What happens when I add or modify a custom word?

How the service responds to a request to add or modify a custom word depends on the fields and values that you specify. It also depends on whether the character sequences of the word exist in the base model's character sequences.

- **Omit both the `sounds_like` and `display_as` fields:**
    - The service sets the `display_as` field to the value of the `word` field. The service does not attempt to set the `sounds_like` field to a pronunciation of the word.
- **Specify only the `sounds_like` field:**
    - *If the `sounds_like` field is valid,* the service sets the value of the `sounds_like` field to the specified value. The service also sets the `display_as` field to the value of the `word` field.
    - *If the `sounds_like` field is invalid:*
        - The `POST /v1/customizations/{customization_id}/words` method adds an `error` field to the word in the model's words resource.
        - The `PUT /v1/customizations/{customization_id}/words/{word_name}` method fails with a 400 response code and an error message. The service does not add the word to the words resource.
- **Specify only the `display_as` field:**
    - The service sets the `display_as` field to the specified value. The service does not attempt to set the `sounds_like` field to a pronunciation of the word.
- **Specify both the `sounds_like` and `display_as` fields:**
    - *If the `sounds_like` field is valid,* the service sets the `sounds_like` and `display_as` fields to the specified values.
    - *If the `sounds_like` field is invalid,* the service responds as it does in the case where the `sounds_like` field is specified but the `display_as` field is not.

Additional transcription efforts

For custom language models that are based on large speech models and next-generation models, the service makes additional efforts to ensure the most effective transcription:

- For sounds-like pronunciations for custom words, the service uses the reverse of the sounds-like as well as its definition in a custom word. For example, given a `sounds_like` field of `I triple E` for the word `IEEE`, the service also effectively uses a reverse sounds-like of `IEEE` for the "word" `I triple E`. This enhances the application of sounds-like pronunciations for speech recognition. (Note that it is not possible for users to create custom words that contain spaces.)
- For acronyms that are parsed from corpora or defined as custom words, the service makes additional speech recognition efforts. An acronym is any word that consists of two or more consecutive uppercase letters. If the acronym contains one or more vowels, the service attempts to recognize the

acronym as a series of individual characters *and* as a pronounced word.

For example, the acronym `NASA` can be read as four individual letters or pronounced as a word that sounds like `nassa`. The service checks for both during speech recognition. This enhances its ability to represent acronyms correctly in a transcript.

Validating a words resource for large speech models and next-generation models

Especially when you add a corpus to a custom language model or add multiple custom words at one time, make sure to perform the following validation:

- *Look for typographical and other errors in corpora.* Especially when you add corpora, which can be large, mistakes are easily made. Make sure to review the corpus closely before adding it to the model.
- *Look for typographical and other errors in custom words.* Make sure to review closely custom words that you add directly to a model.
- *Verify the sounds-like pronunciations.* The service tries to generate sounds-like pronunciations for custom words for which none are specified. In most cases, these pronunciations are sufficient. But the service cannot generate a pronunciation for all words, so you must review the word's definition to ensure that it is complete and valid. Reviewing the pronunciations for accuracy is also recommended for words that have unusual spellings or are difficult to pronounce, and for acronyms and technical terms.

Typographical errors have the unintended consequence of modifying a custom model for nonexistent words, as do ill-formed HTML tags that are left in a corpus file.

To validate and, if necessary, correct a custom word for a custom model, use the following methods:

- List all of the words from a custom model by using the `GET /v1/customizations/{customization_id}/words` method or query an individual word with the `GET /v1/customizations/{customization_id}/words/{word_name}` method. For more information, see Listing custom words from a custom language model.
- Modify words in a custom model to correct errors or to add display-as values by using the `POST /v1/customizations/{customization_id}/words` or `PUT /v1/customizations/{customization_id}/words/{word_name}` method. For more information, see Working with custom words for large speech models and next-generation models.
- Delete extraneous words that are introduced in error (for example, by typographical or other mistakes) by using the `DELETE /v1/customizations/{customization_id}/words/{word_name}` method. For more information, see Deleting a word from a custom language model.

Managing custom language models

The customization interface includes the `POST /v1/customizations` method for creating a custom language model. The interface also includes the `POST /v1/customizations/train` method for training a custom model on the latest data from its words resource. For more information, see

- Create a custom language model
- Train the custom language model

The interface also includes methods for listing information about custom language models, resetting a custom model to its initial state, upgrading a custom model, and deleting a custom model. You cannot train, reset, upgrade, or delete a custom model while the service is handling another operation on that model, including adding resources to the model.

Listing custom language models

The customization interface provides two methods for listing information about the custom language models that are owned by the specified credentials:

- The `GET /v1/customizations` method lists information about all custom language models or, if you specify the `language` parameter, about all custom language models for the specified language. If you specify a language, use the language identifier from the name of the base model, for example, `en-US` for a US English model.
- The `GET /v1/customizations/{customization_id}` method lists information about a specified custom language model. Use this method to poll the service about the status of a training request or a request to add new words.

Both methods return the following information about a custom model:

- `customization_id` identifies the custom model's Globally Unique Identifier (GUID). The GUID is used to identify the model in methods of the interface.
- `created` is the date and time in Coordinated Universal Time (UTC) at which the custom model was created.
- `updated` is the date and time in Coordinated Universal Time (UTC) at which the custom model was last modified.
- `language` is the language of the custom model. The value matches the language identifier from the name of the base model. For example, `en-US` for a US English language model.
- `dialect` is the dialect of the language for the custom model, which does not necessarily match the language of the custom model for previous-generation Spanish models. For more information, see the description of the `dialect` field in Create a custom language model.

- `owner` identifies the credentials of the service instance that owns the custom model.
- `name` is the name of the custom model.
- `description` shows the description of the custom model, if one was provided at its creation.
- `base_model` indicates the name of the language model for which the custom model was created.
- `versions` provides a list of the available versions of the custom model. Each element of the array indicates a version of the base model with which the custom model can be used. Multiple versions exist only if the custom model is upgraded to a new version of its base model. Otherwise, only a single version is shown. For more information, see Listing version information for a custom model.

The method also returns a `status` field that indicates the state of the custom model:

- `pending` indicates that the model was created. It is waiting either for valid training data (corpora, words, or grammars) to be added or for the service to finish analyzing data that was added.
- `ready` indicates that the model contains valid data and is ready to be trained. If the model contains a mix of valid and invalid resources, training of the model fails unless you set the `strict` query parameter to `false`. For more information, see Training failures.
- `training` indicates that the model is being trained on data.
- `available` indicates that the model is trained and ready to use with a recognition request.
- `upgrading` indicates that the model is being upgraded.
- `failed` indicates that training of the model failed. Examine the words in the model's words resource to determine the errors that prevented the model from being trained.

Additionally, the output includes a `progress` field that indicates the status of a custom model's training. If you used the `POST /v1/customizations/{customization_id}/train` method to successfully train the model, this field has a value of `100`. If the model is not completely trained and `available`, the field has a value of `0`.

Tip: When you monitor the training or upgrading of a custom model, poll the value of the `status` field, not the value of the `progress` field. If the operation fails for any reason, the value of the `status` field changes to reflect the failure; the value of the `progress` field remains `0`.

List all custom language models example

The following example includes the `language` query parameter to list all US English custom language models that are owned by the specified credentials:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations?language=en-US"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations?language=en-US"
```

The credentials own two such models. The first model is awaiting data or is being processed by the service. The second model is fully trained and ready for use. Both custom models are based on previous-generation models; the first custom model has two available versions.

```
{
  "customizations": [
    {
      "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96",
      "created": "2016-06-01T14:21:26.894Z",
      "updated": "2020-01-18T18:42:25.324Z",
      "language": "en-US",
      "dialect": "en-US",
      "versions": [
        "en-US_BroadbandModel.v2018-07-31",
        "en-US_BroadbandModel.v2020-01-16"
      ],
      "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
      "name": "Example model",
      "description": "Example custom language model",
      "base_model_name": "en-US_BroadbandModel",
      "status": "pending",
      "progress": 0
    },
    {
      "customization_id": "8391f918-3b76-e109-763c-b7732fae4829",
      "created": "2017-12-02T18:51:37.291Z",
      "updated": "2017-12-02T20:02:10.624Z",
      "language": "en-US",
      "dialect": "en-US",
      "versions": [
        "en-US_BroadbandModel.v2017-11-15"
      ],
      "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
      "name": "Example model two",
      "description": "Example custom language model two",
      "base_model_name": "en-US_BroadbandModel",
      "status": "available",
      "progress": 100
    }
  ]
}
```

List a specific custom language model example

The following example returns information about the custom model that has the specified customization ID:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}"
```

The response duplicates the information from the previous example:

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96",
  "created": "2016-06-01T14:21:26.894Z",
  "updated": "2020-01-18T18:42:25.324Z",
  "language": "en-US",
  "dialect": "en-US",
  "versions": [
    "en-US_BroadbandModel.v2018-07-31",
    "en-US_BroadbandModel.v2020-01-16"
  ],
  "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
  "name": "Example model",
  "description": "Example custom language model",
  "base_model_name": "en-US_BroadbandModel",
  "status": "pending",
  "progress": 0
}
```

Resetting a custom language model

Use the `POST /v1/customizations/{customization_id}/reset` method to reset a custom model. Resetting a model removes all of the corpora and words from the model, initializing the model to its state at creation. The method does not delete the model itself or metadata such as its name and language. However, when you reset a model, its words resource is empty and must be re-created by adding corpora and words.

Reset a custom language model example

The following example resets the custom model with the specified customization ID:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/reset"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/reset"
```

Deleting a custom language model

Use the `DELETE /v1/customizations/{customization_id}` method to delete a custom language model that you no longer need. The method deletes all corpora and words that are associated with the custom model and the model itself. Use this method with caution: a custom model and its data cannot be reclaimed after you delete the model.

Delete a custom language model example

The following example deletes the custom model with the specified customization ID:

IBM Cloud

```
$ curl -X DELETE -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X DELETE \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}"
```

Managing corpora

The customization interface includes the `POST /v1/customizations/{customization_id}/corpora/{corpus_name}` method for adding a corpus to a custom language model. For more information, see Add a corpus to the custom language model. The interface also includes the following methods for listing and deleting corpora for a custom language model.

Listing corpora for a custom language model

The customization interface provides two methods for listing information about the corpora for a custom language model:

- The `GET /v1/customizations/{customization_id}/corpora` method lists information about all corpora for a custom model.
- The `GET /v1/customizations/{customization_id}/corpora/{corpus_name}` method lists information about a specified corpus for a custom model.

Both methods return the `name` of the corpus, the `total_words` read from the corpus, and, *for custom models based on previous-generation models* , the number of `out_of_vocabulary_words` extracted from the corpus. The methods also list the `status` of the corpus. The status is important for checking the service's analysis of a corpus in response to a request to add it to a custom model.

- `analyzed` indicates that the service successfully analyzed the corpus. You can train the custom model with data from the corpus, or you can add additional corpora or words to the model.

- `being_processed` indicates that the service is still analyzing the corpus. The service cannot accept requests to add new corpora or words, or to train the custom model, until its analysis is complete.

- `undetermined` indicates that the service encountered an error while processing the corpus. The information that is returned for the corpus includes an error message that offers guidance for correcting the error.

  For example, the corpus might be invalid, or you might have tried to add a corpus with the same name as an existing corpus. You can try to add the corpus again and include the `allow_overwrite` parameter with the request. You can also delete the corpus and then try adding it again.

List all corpora example

The following example lists all corpora for the custom model with the specified customization ID:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/corpora"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/corpora"
```

Three corpora were added to the custom model. The service successfully analyzed `corpus1`. It is still analyzing `corpus2`, and its analysis of `corpus3` failed. Because the corpora were added to a custom model that is based on a previous-generation model, the `out_of_vocabulary_words` field shows the number of OOV words that the service extracted from the first corpus. The field will show the number of OOV words that are extracted from `corpus2` when it is successfully analyzed. Analysis of `corpus3` failed, so no OOV words were extracted.

```
{
  "corpora": [
    {
      "name": "corpus1",
      "total_words": 5037,
      "out_of_vocabulary_words": 401,
      "status": "analyzed"
    },
    {
      "name": "corpus2",
      "total_words": 0,
      "out_of_vocabulary_words": 0,
      "status": "being_processed"
    },
    {
      "name": "corpus3",
      "total_words": 0,
      "out_of_vocabulary_words": 0,
      "status": "undetermined",
      "error": "Analysis of corpus 'corpus3.txt' failed. Please try adding the corpus again by setting the 'allow_overwrite' flag
to 'true'."
    }
  ]
}
```

List a specific corpus example

The following example returns information about the corpus that is named `corpus1` for the custom model with the specified customization ID:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/corpora/corpus1"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/corpora/corpus1"
```

The corpus, which is based on a previous-generation model, is fully analyzed and contains more than 400 OOV words:

```
{
  "name": "corpus1",
  "total_words": 5037,
  "out_of_vocabulary_words": 401,
  "status": "analyzed"
}
```

Deleting a corpus from a custom language model

Use the `DELETE /v1/customizations/{customization_id}/corpora/{corpus_name}` method to remove an existing corpus from a custom language model.

- *If the custom model is based on a large speech model or next-generation model,* the service deletes the corpus from the model.
- *If the custom model is based on a previous-generation model,* the service deletes the corpus from the model *and* removes OOV words that are

associated with the corpus from the custom model's words resource. The service removes an OOV word from a custom model unless

  - The word was also added by another corpus or by a grammar.
  - The word was modified in some way with the `POST /v1/customizations/{customization_id}/words` or `PUT /v1/customizations/{customization_id}/words/{word_name}` method.

Removing a corpus does not affect the custom model until you train the model on its updated data by using the `POST /v1/customizations/{customization_id}/train` method. If you previously trained the model on the corpus successfully, information extracted from the corpus remains in the model and applies to speech recognition until you retrain the model.

Delete a corpus example

The following example deletes the corpus that is named `corpus3` from the custom model with the specified customization ID:

IBM Cloud

```
$ curl -X DELETE -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/corpora/corpus3"
```

IBM Cloud Pak for Data

```
$ curl -X DELETE \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/corpora/corpus3"
```

Managing custom words

The customization interface includes the `POST /v1/customizations/{customization_id}/words` and `PUT /v1/customizations/{customization_id}/words/{word_name}` methods, which are used to add or modify words for a custom model. For more information, see Add words to the custom language model . The interface also includes the following methods for listing and deleting words for a custom language model.

Character encoding for custom words

In general, you are likely to add most custom words from corpora. Make sure that you know the character encoding that is used in the text files for your corpora. The service preserves the encoding that it finds in the text files.

You must use that same encoding when working with individual words in the custom language model. When you specify a word with the `GET`, `PUT`, or `DELETE /v1/customizations/{customization_id}/words/{word_name}` method, you must URL-encode the `word_name` that you pass in the URL if the word includes non-ASCII characters.

For example, the following table shows what looks like the same letter in two different encodings, ASCII and UTF-8. Although you can pass the ASCII character on a URL as `z`, you must pass the UTF-8 character as `%EF%BD%9A`.

Letter	Encoding	Value
z	ASCII	`0x7a` (`7a`)
ｚ	UTF-8 hexadecimal	`0xEF 0xBD 0x9A` (`efbd9a`)

Table 1. Examples of character encoding

Listing custom words from a custom language model

The customization interface offers two methods for listing words from a custom language model:

- The `GET /v1/customizations/{customization_id}/words` method lists information about the words from the custom model's words resource. The method includes two optional query parameters:

  - The `word_type` parameter specifies which words are to be listed:

    - `all` (the default) shows all words.
    - `user` shows only custom words that were added or modified by the user.
    - `corpora` shows only OOV words that were extracted from corpora.
    - `grammars` shows only OOV words that were extracted from grammars.

    For custom models that are based on next-generation models, only `all` and `user` apply. Both options return the same results. Words from corpora and grammars are not added to the words resource for custom models that are based on next-generation models.

    - The `sort` parameter indicates how the words are to be ordered. The parameter accepts two arguments to indicate how the words are to be sorted: `alphabetical` and `count`. You can add an optional `+` or `-` to the front of an argument to indicate whether the results are to be sorted in ascending or descending order. By default, the method displays the words in ascending alphabetical order.

- The `GET /v1/customizations/{customization_id}/words/{word_name}` method lists information about a single specified word from the model's words resource.

In addition to a `word` field that identifies the word, both methods return the following information about each word:

- A `sounds_like` field that presents an array of as many as five pronunciations for the word. The array of sounds-like pronunciations can include a sounds-like value that is automatically generated by the service if no sounds-like value is provided when the word is added to the custom model. For more information, see
    - *For custom models that are based on previous-generation models,* Using the sounds_like field.
    - *For custom models that are based on next-generation models,* Using the sounds_like field.
- A `display_as` field that shows the spelling of the custom word that the service displays in transcriptions. The field contains an empty string if no display-as value is provided for the word, in which case the word is displayed as it is spelled. For more information, see
    - For custom models that are based on previous-generation models, Using the display_as field.
    - For custom models that are based on next-generation models, Using the display_as field.
- A `source` field that indicates how the word was added to the custom model's words resource.
    - *For custom models that are based on previous-generation models,* the field includes the name of each corpus and grammar from which the service extracted the word. If you modified or added the word directly, the field includes the string `user`.
    - *For custom models that are based on next-generation models,* this field shows only `user` for custom words that were added directly to the custom model. Words from corpora and grammars are not added to the words resource for custom models that are based on next-generation models.
- A `count` field that indicates the number of times the word is found across all corpora and grammars.
    - *For custom models that are based on previous-generation models,* for example, if the word occurs five times in one corpus and seven times in another, its count is `12`. If you add a custom word to a model before it is added by any corpora or grammars, the count begins at `1`. If the word is added from a corpus or grammar first and later modified, the count reflects only the number of times it is found in corpora and grammars.
    - *For custom models that are based on large speech models and next-generation models,* the `count` field for any word is always `1`.

If the service discovers one or more problems with a custom word's definition, the output includes an `error` field. The field provides an array that lists each problem element from the definition and a message that describes the problem.

An error can occur, for example, if you add a custom word with an invalid `sounds_like` field, one that violates one of the rules for adding a pronunciation. You cannot train a custom model whose words resource includes a word with an error. You must correct or delete the word before you can train the model.

List all custom words example

The following example lists all of the words, regardless of type, from the custom model with the specified customization ID. The words are displayed in the default sort order, ascending alphabetical.

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/words"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/words"
```

The words resource for the model contains four words. The first word was added directly by the user, but its `sounds_like` field contains an error: The field cannot contain numbers. The other words were added by the user or by both the user and from corpora, which indicates that the words were added to a custom model that is based on a previous-generation model.

```
{
  "words": [
    {
      "word": "75.00",
      "sounds_like": ["75 dollars"],
      "display_as": "75.00",
      "count": 1,
      "source": ["user"],
      "error": [{"75 dollars": "Numbers are not allowed in sounds_like. You can try for example 'seventy five dollars'."}]
    },
    {
      "word": "HHonors",
      "sounds_like": [
        "hilton honors",
        "H. honors"
      ],
      "display_as": "HHonors",
      "count": 1,
      "source": [
        "corpus1",
        "user"
      ]
    },
    {
      "word": "IEEE",
      "sounds_like": ["I. triple E."],
      "display_as": "IEEE",
      "count": 3,
      "source": [
        "corpus1",
        "corpus2",
        "user"
      ]
    },
    {
      "word": "tomato",
      "sounds_like": [
        "tomatoh",
        "tomayto"
      ],
      "display_as": "tomato",
      "count": 1,
      "source": ["user"]
    }
  ]
}
```

List a specific custom word example

The following example shows information about the word `NCAA` from the words resource of the specified model:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/words/NCAA"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/words/NCAA"
```

The user added the word initially. The service then found the word twice in `corpus3`. These words were also added to a custom model that is based on a previous-generation model.

```
{
  "word": "NCAA",
  "sounds_like": [
    "N. C. A. A.",
    "N. C. double A."
  ],
  "display_as": "NCAA",
  "count": 3,
  "source": [
    "corpus3",
    "user"
  ]
}
```

Deleting a custom word from a custom language model

Use the `DELETE /v1/customizations/{customization_id}/words/{word_name}` method to delete a word from a custom language model.

- *For custom models that are based on large speech models, previous-generation models,* use the method to remove words that were added in error, for example, from a corpus with faulty data. You can remove any word that you added to the custom model's words resource via any means (for example, from corpora, from grammars, or directly.)
- *For custom models that are based on next-generation models,* you can delete only words that were added directly to the model. Words are not added to the model from corpora or grammars.

You cannot delete a word from the service's base vocabulary. If you delete a word that is also present in the service's base vocabulary, deleting the word from a custom model deletes only the custom pronunciation for the word. The word remains in the base vocabulary.

Removing a word from a custom model does not affect the model until you retrain it by using the `POST /v1/customizations/{customization_id}/train` method. If the model was previously trained on the word, the model continues to apply the word to speech recognition even after you delete the word from its words resource. You must retrain the model to reflect the deletion.

Delete a custom word example

The following example deletes the word `IEEE` from the custom model with the specified customization ID:

IBM Cloud

```
$ curl -X DELETE -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/words/IEEE"
```

IBM Cloud Pak for Data

```
$ curl -X DELETE \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/words/IEEE"
```

Acoustic model customization

Creating a custom acoustic model

Note: Acoustic model customization is available only for previous-generation models. It is not available for next-generation and large speech models.

Follow these steps to create a custom acoustic model for the IBM Watson® Speech to Text service:

1. Create a custom acoustic model. You can create multiple custom models for the same or different domains or environments. The process is the same for any model that you create. Acoustic model customization is available for all languages that are supported by previous-generation models. For more information about which languages are generally available and beta, see Language support for customization.

2. Add audio to the custom acoustic model. The service accepts the same audio file formats for acoustic modeling that it accepts for speech recognition. It also accepts archive files that contain multiple audio files. Archive files are the preferred means of adding audio resources. You can repeat the method to add more audio or archive files to a custom model.

3. Train the custom acoustic model. Once you add audio resources to the custom model, you must train the model. Training prepares the custom acoustic model for use in speech recognition. The training time depends on the cumulative amount of audio data that the model contains.

   You can specify a helper custom language model during training of your custom acoustic model. A custom language model that includes transcriptions of your audio files or OOV words from the domain of your audio files can improve the quality of the custom acoustic model. For more information, see Training a custom acoustic model with a custom language model.

4. After you train your custom model, you can use it with recognition requests. If the audio passed for transcription has acoustic qualities that are similar to the audio of the custom model, the results reflect the service's enhanced understanding. You can use only one custom acoustic model at a time with a speech recognition request. For more information, see Using a custom acoustic model for speech recognition.

   You can pass both a custom acoustic model and a custom language model in the same recognition request to further improve recognition accuracy. For more information, see Using custom language and custom acoustic models for speech recognition.

The steps for creating a custom acoustic model are iterative. You can add or delete audio and retrain a model as often as needed. You must retrain a model for any changes to its audio to take effect.

Create a custom acoustic model

You use the `POST /v1/acoustic_customizations` method to create a new custom acoustic model. The method accepts a JSON object that defines the attributes of the new custom model as the body of the request. The new custom model is owned by the instance of the service whose credentials are used to create it. For more information, see Ownership of custom models.

You can create a maximum of 1024 custom acoustic models per owning credentials. For more information, see Maximum number of custom models.

A new custom acoustic model has the following attributes:

`name` (*required* string)

A user-defined name for the new custom acoustic model. Use a localized name that matches the language of the custom model and reflects the acoustic environment of the model, such as `Mobile custom model` or `Noisy car custom model`.

- Include a maximum of 256 characters in the name.
- Do not use backslashes, slashes, colons, equal signs, ampersands, or question marks in the name.
- Use a name that is unique among all custom acoustic models that you own.

`base_model_name` (*required* string)

The name of the base language model that is to be customized by the new model. You must use the name of a model that is returned by the `GET /v1/models` method. The new custom model can be used only with the base model that it customizes.

`description` (*optional* string)

A recommended description of the new custom model.

- Use a localized description that matches the language of the custom model.
- Include a maximum of 128 characters in the description.

The following example creates a new custom acoustic model named `Example acoustic model`. The model is created for the base model `en-US_BroadbandModel` and has the description `Example custom acoustic model`. The `Content-Type` header specifies that JSON data is being passed to the method.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: application/json" \
--data "{\"name\": \"Example acoustic model\", \
  \"base_model_name\": \"en-US_BroadbandModel\", \
  \"description\": \"Example custom acoustic model\"}" \
"{url}/v1/acoustic_customizations"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/json" \
--data "{\"name\": \"Example acoustic model\", \
  \"base_model_name\": \"en-US_BroadbandModel\", \
  \"description\": \"Example custom acoustic model\"}" \
"{url}/v1/acoustic_customizations"
```

The example returns the customization ID of the new model. Each custom model is identified by a unique customization ID, which is a Globally Unique Identifier (GUID). You specify a custom model's GUID with the `customization_id` parameter of calls that are associated with the model.

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96"
}
```

Add audio to the custom acoustic model

Once you create your custom acoustic model, the next step is to add audio resources to it. You use the `POST /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method to add an audio resource to a custom model. You can add

- An individual audio file in any format that is supported for speech recognition. For more information, see Supported audio formats.
- An archive file (a `.zip` or `.tar.gz` file) that includes multiple audio files. Gathering multiple audio files into a single archive file and loading that single file is significantly more efficient than adding audio files individually.

You pass the audio resource as the body of the request and assign the resource an `audio_name`. For more information, see Working with audio resources.

The following examples show the addition of both audio- and archive-type resources:

- This example adds an audio-type resource to the custom acoustic model with the specified `customization_id`. The `Content-Type` header identifies the type of the audio as `audio/wav`. The audio file, `audio1.wav`, is passed as the body of the request, and the resource is given the name `audio1`.

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  --header "Content-Type: audio/wav" \
  --data-binary @audio1.wav \
  "{url}/v1/acoustic_customizations/{customization_id}/audio/audio1"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X POST \
  --header "Authorization: Bearer {token}" \
  --header "Content-Type: audio/wav" \
  --data-binary @audio1.wav \
  "{url}/v1/acoustic_customizations/{customization_id}/audio/audio1"
  ```

- This example adds an archive-type resource to the specified custom acoustic model. The `Content-Type` header identifies the type of the archive as `application/zip`. The `Contained-Contented-Type` header indicates that all files that are contained in the archive have the format `audio/l16` and are sampled at a rate of 16 kHz. The archive file, `audio2.zip`, is passed as the body of the request, and the resource is given the name `audio2`.

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  --header "Content-Type: application/zip" \
  --header "Contained-Content-Type: audio/l16;rate=16000" \
  --data-binary @audio2.zip \
  "{url}/v1/acoustic_customizations/{customization_id}/audio/audio2"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X POST \
  --header "Authorization: Bearer {token}" \
  --header "Content-Type: application/zip" \
  --header "Contained-Content-Type: audio/l16;rate=16000" \
  --data-binary @audio2.zip \
  "{url}/v1/acoustic_customizations/{customization_id}/audio/audio2"
  ```

The method also accepts an optional `allow_overwrite` query parameter to overwrite an existing audio resource for a custom model. Use the parameter if you need to update an audio resource after you add it to a model.

The method is asynchronous. It can take several seconds or minutes to complete depending on the duration of the audio. For an archive file, the length of the operation depends on the duration of all audio files contained in the archive. The duration of the operation also depends on the current load on the service. For more information about checking the status of a request to add an audio resource, see Monitoring the add audio request.

You can add any number of audio resources to a custom model by calling the method once for each audio or archive file. You can make multiple requests to add different audio resources simultaneously.

You must add a minimum of 10 minutes of audio that includes speech, not silence, to a custom acoustic model before you can train it. No audio- or archive-type resource can be larger than 100 MB.

- For general guidance about adding audio to a custom acoustic model, see Guidelines for adding audio.
- For more information about the maximum amount of audio that you can add to a custom acoustic model, see Maximum hours of audio .

Monitoring the add audio request

The service returns a 201 response code if the audio is valid. It then asynchronously analyzes the contents of the audio file or files and automatically extracts information about the audio such as length, sampling rate, and encoding. You cannot train the custom model until the service's analysis of all audio resources for current requests completes.

To determine the status of the request, use the `GET /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method to poll the status of the audio. The method accepts the customization ID of the custom model and the name of the audio resource. Its response includes the `status` of the resource, which has one of the following values:

- `ok` indicates that the audio is acceptable and analysis is complete.
- `being_processed` indicates that the service is still analyzing the audio.
- `invalid` indicates that the audio file is not acceptable for processing. It might have the wrong format, the wrong sampling rate, or not be an audio file. For an archive file, if any of the audio files that it contains are invalid, the entire archive is invalid.

The content of the response and location of the `status` field depend on the type of the resource, audio or archive.

- *For an audio-type resource,* the `status` field is located in the top-level (`AudioListing`) object.

  IBM Cloud

  ```
  $ curl -X GET -u "apikey:{apikey}" \
  "{url}/v1/acoustic_customizations/{customization_id}/audio/audio1"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X GET \
  --header "Authorization: Bearer {token}" \
  "{url}/v1/acoustic_customizations/{customization_id}/audio/audio1"
  ```

  The status of the audio-type resource is `ok`:

  ```
  {
    "duration": 131,
    "name": "audio1",
    "details": {
      "codec": "pcm_s16le",
      "type": "audio",
      "frequency": 22050
    }
    "status": "ok"
  }
  ```

- *For an archive-type resource,* the `status` field is located in the second-level (`AudioResource`) object that is nested in the `container` field.

  IBM Cloud

  ```
  $ curl -X GET -u "apikey:{apikey}" \
  "{url}/v1/acoustic_customizations/{customization_id}/audio/audio2"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X GET \
  --header "Authorization: Bearer {token}" \
  "{url}/v1/acoustic_customizations/{customization_id}/audio/audio2"
  ```

  The status of the archive-type resource is `ok`:

```
{
  "container": {
    "duration": 556,
    "name": "audio2",
    "details": {
      "type": "archive",
      "compression": "zip"
    },
    "status": "ok"
  },
  . . .
}
```

Use a loop to check the status of the audio resource every few seconds until it becomes `ok`. For more information about other fields that are returned by the method, see Listing audio resources for a custom acoustic model.

Train the custom acoustic model

Once you populate a custom acoustic model with audio resources, you must train the model on the new data. Training prepares a custom model for use in speech recognition. A model cannot be used for recognition requests until you train it on the new data. Also, updates to a custom model in the form of new or changed audio resources are not reflected by the model until you train it with the changes.

You use the `POST /v1/acoustic_customizations/{customization_id}/train` method to train a custom model. You pass the method the customization ID of the model that you want to train, as in the following example.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}/train"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}/train"
```

The method is asynchronous. Training time depends on the cumulative amount of audio data that the custom acoustic model contains and the current load on the service. When you train or retrain a model, the service uses all of the model's audio data (not just new data) in the training. So training time is commensurate with the total amount of audio that the model contains.

A general guideline is that training a custom acoustic model takes approximately as long as the length of its cumulative audio data. For example, it takes approximately 2 hours to train a model that contains a total of 2 hours of audio. For more information about checking the status of a training operation, see Monitoring the train model request.

The method includes the following optional query parameters:

- The `custom_language_model_id` parameter specifies a separately created custom language model that is to be used during training. You can train with a custom language model that contains transcriptions of your audio files or that contains corpora or OOV words that are relevant to the contents of the audio files. For training to succeed, the custom language model must be fully trained and available, and the custom acoustic and custom language models must be based on the same version of the same base model. For more information, see Training a custom acoustic model with a custom language model.
- The `strict` parameter indicates whether training is to proceed if the custom model contains a mix of valid and invalid audio resources. By default, training fails if the model contains one or more invalid resources. Set the parameter to `false` to allow training to proceed as long as the model contains at least one valid resource. The service excludes invalid resources from the training. For more information, see Training failures for custom acoustic models.

Monitoring the train model request

The service returns a 200 response code if the training process is successfully initiated. The service cannot accept subsequent training requests, or requests to add more audio resources, until the existing training request completes.

To determine the status of a training request, use the `GET /v1/acoustic_customizations/{customization_id}` method to poll the model's status. The method accepts the customization ID of the acoustic model, as in the following example:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}"
```

The response includes the status of the model, which is `training`:

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96",
  "created": "2016-06-01T18:42:25.324Z",
  "updated": "2016-06-01T22:11:13.298Z",
  "language": "en-US",
  "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
  "name": "Example model",
  "description": "Example custom acoustic model",
  "base_model_name": "en-US_BroadbandModel",
  "status": "training",
  "progress": 0
}
```

The response includes `status` and `progress` fields that report the current state of the model. The meaning of the `progress` field depends on the model's status. The `status` field can have one of the following values:

- `pending` indicates that the model was created but is waiting either for valid training data to be added or for the service to finish analyzing data that was added. The `progress` field is `0`.
- `ready` indicates that the model contains valid data and is ready to be trained. The `progress` field is `0`.

  If the model contains a mix of valid and invalid audio resources, training of the model fails unless you set the `strict` query parameter to `false`. For more information, see Training failures for custom acoustic models.
- `training` indicates that the model is being trained. The `progress` field changes from `0` to `100` when training is complete.
- `available` indicates that the model is trained and ready to use. The `progress` field is `100`.
- `upgrading` indicates that the model is being upgraded. The `progress` field is `0`.
- `failed` indicates that training of the model failed. The `progress` field is `0`. For more information, see Training failures for custom acoustic models.

Use a loop to check the status of the training once a minute until the model becomes `available`. For more information about other fields that are returned by the method, see Listing custom acoustic models.

Training failures for custom acoustic models

Training fails to start if the service is handling another request for the custom acoustic model. A conflicting request could be another training request or a request to add audio resources to the model. The service returns a status code of 409.

Training also fails to start for the following reasons:

- The custom model contains less than 10 minutes of audio data.
- The custom model contains more than 200 hours of audio data.
- One or more of the custom model's audio resources is invalid.
- You passed a custom language model with the `custom_language_model_id` query parameter that is not in the `available` state. A custom language model must be fully trained and available to be used to train a custom acoustic model.
- You passed an incompatible custom language model with the `custom_language_model_id` query parameter. Both custom models must be based on the same version of the same base model.

The service returns a status code of 400 and sets the custom model's status to `failed`. Take one of the following actions:

- Use the `GET /v1/acoustic_customizations/{customization_id}/audio` and `GET /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` methods to examine the model's audio resources. For more information, see Listing audio resources for a custom acoustic model.

For each invalid audio resource, do one of the following:

- Correct the audio resource and use the `allow_overwrite` parameter of the `POST /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method to add the corrected audio to the model. For more information, see Add audio to the custom acoustic model.
- Use the `DELETE /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method to delete the audio resource from the model. For more information, see Deleting an audio resource from a custom acoustic model.

- Set the `strict` parameter of the `POST /v1/acoustic_customizations/{customization_id}/train` method to `false` to exclude invalid audio resources from the training. The model must contain at least one valid audio resource for training to succeed. The `strict` parameter is useful for training a custom model that contains a mix of valid and invalid audio resources.

Using a custom acoustic model for speech recognition

Note: Acoustic model customization is available only for previous-generation models. It is not available for next-generation and large speech models.

Once you create and train your custom acoustic model, you can use it in speech recognition requests by using the `acoustic_customization_id` query parameter. By default, no custom acoustic model is used with a request. You can create multiple custom acoustic models for the same or different domains. But you can specify only one custom acoustic model at a time for a speech recognition request. You must issue the request with credentials for the instance of the service that owns the custom model.

A custom model can be used only with the base model for which it is created. If your custom model is based on a model other than the default, you must also specify that base model with the `model` query parameter. For more information, see Using the default model.

You can also specify a custom language model to be used with the request, which can increase transcription accuracy. For more information, see Using custom language and custom acoustic models for speech recognition.

Examples of using a custom acoustic model

The following examples show the use of a custom acoustic model with each speech recognition interface:

- For the WebSocket interface, use the `/v1/recognize` method. The specified custom model is used for all requests that are sent over the connection.

```
var access_token = {access_token};
var wsURI = '{ws_url}/v1/recognize'
  + '?access_token=' + access_token
  + '&model=en-US_NarrowbandModel'
  + '&acoustic_customization_id={customization_id}';
var websocket = new WebSocket(wsURI);
```

- For the synchronous HTTP interface, use the `POST /v1/recognize` method. The specified custom model is used for that request.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file1.flac \
"{url}/v1/recognize?acoustic_customization_id={customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file1.flac \
"{url}/v1/recognize?acoustic_customization_id={customization_id}"
```

- For the asynchronous HTTP interface, use the `POST /v1/recognitions` method. The specified custom model is used for that request.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognitions?acoustic_customization_id={customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file.flac \
"{url}/v1/recognitions?acoustic_customization_id={customization_id}"
```

You can omit the language model from the request if the custom model is based on the default model, `en-US_BroadbandModel`. Otherwise, you must use the `model` parameter to specify the base model, as shown for the WebSocket example. A custom model can be used only with the base model for which it is created.

Troubleshooting the use of custom acoustic models

If you apply a custom acoustic model to speech recognition but find that the quality of speech recognition does not improve, check for the following possible problems:

- Make sure that you are correctly passing the customization ID to the recognition request as shown in the previous examples.
- Make sure that the status of the custom model is `available`, meaning that it is fully trained and ready to use. For more information, see Listing custom acoustic models.

Using custom acoustic and custom language models together

Note: Acoustic model customization is available only for previous-generation models. It is not available for next-generation and large speech models.

You can improve speech recognition accuracy by using complementary custom language and custom acoustic models. You can use both types of model during training of your acoustic model, during speech recognition, or both. The custom language and custom acoustic models must be owned by the same service instance, and both must customize the same base language model.

Training a custom acoustic model with a custom language model

The following terms differentiate how a custom acoustic model is trained, alone or with a custom language model:

- *Unsupervised training* refers to training a custom acoustic model with audio data alone. Training a custom acoustic model on audio alone can improve transcription quality if the characteristics of the custom model's audio match those of the audio that is being transcribed.
- *Lightly supervised training* refers to training a custom acoustic model with a complementary, or "helper," custom language model. Lightly supervised training is more effective than unsupervised training in improving the quality of speech recognition.

Use lightly supervised training in the following cases:

- When you have a custom language model that contains transcripts of your audio files.

  Transcribing your audio data is not strictly necessary. But training with a custom language model whose corpora are based on transcripts of your audio files can improve speech recognition. This is especially true if the audio data contains out-of-vocabulary (OOV) words that are not found in the service's base vocabulary. The service can parse the contents of the transcriptions in context, extracting OOV words and n-grams that can help it make the most effective use of your audio data.

- When you have a custom language model that is based on corpora or words that are relevant to the contents of your audio files.

  If you don't have transcripts of your audio files, you can train with a custom language model that includes OOV words from the same domain as your audio data. Training with a custom language model that includes OOV words that are used in the audio can improve speech recognition.

For example, suppose you are creating a custom acoustic model that is based on call-center audio for specific products. Optimally, you can train the custom acoustic model with a custom language model that contains transcripts of related calls. If transcripts are unavailable, you can still train with a custom language model that includes just names of specific products that are handled by the call center.

Guidelines for using lightly supervised training

Follow these guidelines when you have transcripts of the audio files from the custom acoustic model:

- Create a dedicated custom language model whose sole purpose is to help train this specific custom acoustic model. The dedicated helper model can contain multiple transcripts for multiple audio files of the same custom acoustic model. It can also include custom words that are relevant to the audio files, though transcripts are more effective in training the custom acoustic model.

- Use the dedicated custom language model solely for training the custom acoustic model. Once the custom acoustic model is trained, the custom language model needs to be used only if you update the audio files in the acoustic model.
- Add transcripts of your audio files to the custom language model as corpora. It is not strictly necessary for a corpus to include only one sentence per line. But as with all corpora, the service makes noticeably better use of a corpus that includes each sentence on its own line. In general, shorter utterances that occupy separate lines of the corpus are most effective.
- It is neither necessary nor possible to associate a specific corpora with a specific audio file. A custom language model can contain multiple corpora, just as a custom acoustic model can include multiple audio files. All contents of a custom language model help improve the internal transcript of the audio that the service generates during the training process.
- A transcript does not need to be a verbatim reflection of all sentences and words from the audio. It can include only sentences and words of the audio that are relevant to the domain. For both language model and acoustic model customization, the training data needs to reflect the actual use-case of the speech that you want to recognize. If only 20 percent of the data (transcript or audio) is specific to the domain of the use-case, use just that 20 percent of the data for training.

  However, if you are using acoustic model customization to achieve better results for accented speech (for example, for speech by non-native speakers), keep as much audio as possible, even if it's not relevant to the domain. The same is true if you are using acoustic model customization to improve speech recognition accuracy under difficult acoustic conditions, such as a noisy background.
- A verbatim transcript does not need to contain the disfluencies, verbal tics, and filler statements that are common to human speech. You can remove these elements from the transcript, the audio, or both. (You can also remove audio in which people are talking over each other, since that does not contribute to the training.)

  For example, suppose a verbatim transcript includes the following sentence:

  `So that's, uhm, you know, that, as I say, is the is the predominant form today.`

  You can edit these peculiarities from the transcript, the audio, or both to create the following sentence:

  `So that as I say is the predominant form today.`

Performing lightly supervised training

To train a custom acoustic model with a custom language model, you use the optional `custom_language_model_id` query parameter of the `POST /v1/acoustic_customizations/{customization_id}/train` method. Pass the GUID of the acoustic model with the `customization_id` parameter and the GUID of the custom language model with the `custom_language_model_id` parameter. Both models must be owned by the credentials that are passed with the request.

- Ensure that the custom language model is fully trained and in the `available` state. Training fails if the custom language model is not `available`.
- Ensure that both custom models are based on the same version of the same base model. If a new version of the base model is made available, you must upgrade both models to the same version of the base model for training to succeed. For more information, see Upgrading custom models.

Lightly supervised training example

The following example request shows a custom language model being used to train a custom acoustic model:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}/train?custom_language_model_id={customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}/train?custom_language_model_id={customization_id}"
```

Using custom language and custom acoustic models for speech recognition

You can specify both a custom language model and a custom acoustic model with any speech recognition request. If your audio contains domain-specific OOV words, acoustic model customization alone cannot reliably produce those words during speech recognition. Using a custom language model that contains OOV words from the domain of the audio is the only way to expand the service's base vocabulary.

Using a custom language model can improve transcription accuracy regardless of whether you trained the custom acoustic model with the custom language model:

- Using both custom language and custom acoustic models during training improves the quality of the custom acoustic model.
- Using both types of model during speech recognition improves transcription quality.

If a custom language model includes grammars, you can also use the custom language model and one of its grammars with a custom acoustic model during speech recognition.

For a speech recognition request, use the `acoustic_customization_id` and `language_customization_id` parameters to pass the GUIDs of the custom acoustic and custom language models for the request. Both custom models must be owned by the credentials that are passed with the request, both must be based on the same base model (for example, `en-US_BroadbandModel`), and both must be in the `available` state.

Note: For more information about the effects of custom model upgrading on speech recognition with both types of custom models, see Considerations for using custom acoustic and custom language models together .

Example of using both custom language and custom acoustic models

The following example request passes both custom acoustic and custom language models to the HTTP `POST /v1/recognize` method:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file1.flac \
"{url}/v1/recognize?acoustic_customization_id={customization_id}&language_customization_id={customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @audio-file1.flac \
"{url}/v1/recognize?acoustic_customization_id={customization_id}&language_customization_id={customization_id}"
```

For an asynchronous HTTP request, you specify the parameters when you create the asynchronous job. For a WebSocket request, you pass the parameters when you establish a connection. For more information, see Using a custom language model for speech recognition and Using a custom acoustic model for speech recognition.

Working with audio resources

Note: Acoustic model customization is available only for previous-generation models. It is not available for next-generation and large speech models.

You can add individual audio files or archive files that contain multiple audio files to a custom acoustic model. The recommended means of adding audio resources is by adding archive files. Creating and adding a single archive file is considerably more efficient than adding multiple audio files individually. You can also submit requests to add multiple different audio resources at the same time.

Adding an audio resource

You use the `POST /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method to add either type of audio resource to a custom acoustic model. You pass the audio resource as the body of the request and include the following parameters:

- The `customization_id` path parameter to specify the customization ID of the model.
- The `audio_name` path parameter to specify a name for the audio resource.
  - Use a localized name that matches the language of the custom model and reflects the contents of the resource.
  - Include a maximum of 128 characters in the name.
  - Do not use characters that need to be URL-encoded. For example, do not use spaces, slashes, backslashes, colons, ampersands, double quotes, plus signs, equals signs, questions marks, and so on in the name. (The service does not prevent the use of these characters. But because they must be URL-encoded wherever used, their use is strongly discouraged.)
  - Do not use the name of an audio resource that has already been added to the custom model.

When you update a model's audio resources, you must train the model for the changes to take effect during transcription. For more information, see Train the custom acoustic model.

Adding an audio file

To add an individual audio file to a custom acoustic model, you specify the format (MIME type) of the audio with the `Content-Type` header. You can add audio with any format that is supported for use with recognition requests. Include the `rate`, `channels`, and `endianness` parameters with the specification of formats that require them. For more information, see Supported audio formats.

Note: The `application/octet-stream` specification for an audio format is not supported for audio resources.

The following example from Add audio to the custom acoustic model adds an `audio/wav` file:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/wav" \
--data-binary @audio1.wav \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio1"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/wav" \
--data-binary @audio1.wav \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio1"
```

Adding an archive file

The preferred means of adding audio to a custom acoustic model is to add an archive file that includes multiple audio files. You can add the following types of archive files by specifying the type of the archive with the `Content-Type` request header:

- A `.zip` file by specifying `application/zip`
- A `.tar.gz` file by specifying `application/gzip`

You might also need to specify the `Contained-Content-Type` header depending on the format of the files that you are adding:

- For audio files of type `audio/alaw`, `audio/basic`, `audio/l16`, or `audio/mulaw`, you must use the `Contained-Content-Type` header to specify the format of the audio files. Include the `rate`, `channels`, and `endianness` parameters where necessary. In this case, all audio files contained in the archive file must have the same audio format.
- For audio files of all other types, you can omit the `Contained-Content-Type` header. In this case, the audio files contained in the archive file can have any of the formats not listed in the previous bullet. They do not need to have the same format.

Note: Do not use the `Contained-Content-Type` header when adding an audio-type resource.

The name of an audio file that is contained in an archive-type resource can include a maximum of 128 characters. This includes the file extension and all elements of the name (for example, slashes).

The following example from Add audio to the custom acoustic model adds an `application/zip` file that contains audio files in `audio/l16` format that are sampled at 16 kHz:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: application/zip" \
--header "Contained-Content-Type: audio/l16;rate=16000" \
--data-binary @audio2.zip \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio2"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/zip" \
--header "Contained-Content-Type: audio/l16;rate=16000" \
--data-binary @audio2.zip \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio2"
```

Guidelines for adding audio

The improvement in recognition accuracy that you can expect from using a custom acoustic model depends on a number of factors. These factors include how much audio data the custom acoustic model contains and how similar that data is to the audio that is being transcribed. The improvement also depends on whether the custom acoustic model is trained with a corresponding custom language model.

Follow these guidelines when you add audio resources to a custom acoustic model:

- Add at least 10 minutes of audio that includes speech, not silence.

  The quality of the audio makes a difference when you are determining how much to add. The better the model's audio reflects the characteristics of the audio that is to be recognized, the better the quality of the custom model for speech recognition. If the audio is of good quality, adding more can improve transcription accuracy. But adding even five to ten hours of good quality audio can make a positive difference.

- Add audio resources that are no larger than 100 MB. All audio- and archive-type resources are limited to a maximum size of 100 MB.

  To maximize the amount of audio that you can add with a single resource, consider using an audio format that offers compression. For more information, see [Data limits and compression](#).

- Divide large audio files into multiple smaller files. Make sure to split the audio between words, at points of silence.

  Because you can submit multiple simultaneous requests to add different audio resources, you can add smaller files concurrently. This parallel approach to adding audio resources can accelerate the service's analysis of your audio.

- Add audio content that reflects the acoustic channel conditions of the audio that you plan to transcribe. For example, if your application deals with audio that has background noise from a moving vehicle, use the same type of data to build the custom model.

- Make sure that the sampling rate of an audio file matches the sampling rate of the base model for the custom acoustic model:

  - For broadband models, the sampling rate must be at least 16 kHz (16,000 samples per second).
  - For narrowband models, the sampling rate must be at least 8 kHz (8000 samples per second).

  If the sampling rate of the audio is higher than the minimum required sampling rate, the service downsamples the audio to the appropriate rate. If the sampling rate of the audio is lower than the minimum required rate, the service labels the audio file as `invalid`. If any audio file that is contained in an archive file is invalid, the service considers the entire archive invalid.

- Create a custom language model to use with your custom acoustic model in the following cases:

  - If your audio is less than an hour long, create a custom language model based on transcriptions of the audio to achieve the best results.
  - If your audio is domain-specific and contains unique words that are not found in the service's base vocabulary, use language model customization to expand the service's base vocabulary. Acoustic model customization alone cannot produce those words during transcription.

  For more information, see [Using custom acoustic and custom language models together](#).

Maximum hours of audio

The maximum hour of audio that you can add to a custom acoustic model depend on the version of the service that you are using:

- IBM Cloud Pak for Data Add a maximum of 200 hours of audio data.

- IBM Cloud Add a maximum of approximately 50 hours of audio data. You might be able to add 10 or 20 hours of additional audio data per model. However, it is safer to assume a maximum of 50 hours of audio data per custom acoustic model and plan accordingly.

  The maximum amount of audio per custom acoustic model was previously 200 hours. The limit is being reduced by location according to the schedule in Table 1.

Location	Planned date of new limit	Actual date of new limit
`eu-gb` (London)	11 August 2022	11 August 2022
`au-syd` (Sydney)	19 October 2022	19 October 2022
`us-east` (Washington, DC)	First-quarter 2023	TBD
`us-south` (Dallas)	First-quarter 2023	TBD
`eu-de` (Frankfurt)	First-quarter 2023	TBD

`jp-tok` (Tokyo)	First-quarter 2023	TBD

Table 1. Schedule of 50-hour limit per location

If you currently have a custom acoustic model with more than 50 hours of audio data, you can do one of the following:

- Leave the model alone. The model will continue to work with speech recognition requests. However, you cannot add more audio to the model or retrain the model.
- Reduce the amount of audio that the model contains. Maintain the audio that makes the most difference, meaning the audio that is most representative of the characteristics that you wish to address with the model. Remove audio that is less characteristics of the audio you plan to recognize. By using only better quality audio, you can reduce the amount audio the you need for the model.
- If possible, separate the audio into multiple models. Create new, smaller models that address specific characteristics that the original model addressed. For example, you might be using the same model to represent multiple audio characteristics. Instead of relying on a single large model, create multiple smaller models that represent more refined characteristics. This might not be possible for all models. But because you can specify only a single custom acoustic model with a speech recognition request, it can help if your data accommodates such refinement.

Managing custom acoustic models

Note: Acoustic model customization is available only for previous-generation models. It is not available for next-generation and large speech models.

The customization interface includes the `POST /v1/acoustic_customizations` method for creating a custom acoustic model. The interface also includes the `POST /v1/acoustic_customizations/train` method for training a custom model on its latest audio resources. For more information, see

- Create a custom acoustic model
- Train the custom acoustic model

In addition, the interface includes methods for listing information about custom acoustic models, resetting a custom model to its initial state, upgrading a custom model, and deleting a custom model. You cannot train, reset, upgrade, or delete a custom model while the service is handling another operation on that model, including adding audio resources to the model.

Listing custom acoustic models

The customization interface provides two methods for listing information about the custom acoustic models that are owned by the specified credentials:

- The `GET /v1/acoustic_customizations` method lists information about all custom acoustic models or about all custom acoustic models for a specified language.
- The `GET /v1/acoustic_customizations/{customization_id}` method lists information about a specified custom acoustic model. Use this method to poll the service about the status of a training request.

Both methods return the following information about a custom acoustic model:

- `customization_id` identifies the custom model's Globally Unique Identifier (GUID). The GUID is used to identify the model in methods of the interface.
- `created` is the date and time in Coordinated Universal Time (UTC) at which the custom model was created.
- `updated` is the date and time in Coordinated Universal Time (UTC) at which the custom model was last modified.
- `language` is the language of the custom model.
- `owner` identifies the credentials of the service instance that owns the custom model.
- `name` is the name of the custom model.
- `description` shows the description of the custom model, if one was provided at its creation.
- `base_model_name` indicates the name of the language model for which the custom model was created.
- `versions` provides a list of the available versions of the custom model. Each element of the array indicates a version of the base model with which the custom model can be used. Multiple versions exist only if the custom model is upgraded to a new version of its base model. Otherwise, only a single version is shown. For more information, see Listing version information for a custom model.

The methods also return a `status` field that indicates the state of the custom model:

- `pending` indicates that the model was created. It is waiting either for valid training data (audio resources) to be added or for the service to finish analyzing data that was added.
- `ready` indicates that the model contains valid audio data and is ready to be trained. If the model contains a mix of valid and invalid audio resources, training of the model fails unless you set the `strict` query parameter to `false`. For more information, see Training failures.
- `training` indicates that the model is being trained on audio data.

- `available` indicates that the model is trained and ready to use with recognition requests.
- `upgrading` indicates that the model is being upgraded.
- `failed` indicates that training of the model failed. Examine the model's audio resources to determine what prevented the model from being trained. Possible errors include not enough audio, too much audio, or an invalid audio resource.

Additionally, the output includes a `progress` field that indicates the current progress of the custom model's training. If you used the `POST /v1/acoustic_customizations/{customization_id}/train` method to start training the model, this field indicates the current progress of that request as a percentage complete. At this time, the value of the field is `100` if the status is `available`; otherwise, it is `0`.

Tip: When you monitor the training or upgrading of a custom model, poll the value of the `status` field, not the value of the `progress` field. If the operation fails for any reason, the value of the `status` field changes to reflect the failure; the value of the `progress` field remains `0`.

List all custom acoustic models example

The following example includes the `language` query parameter to list all US English custom acoustic models that are owned by the specified credentials:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations?language=en-US"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations?language=en-US"
```

The credentials own two such models. The first model is awaiting data or is being processed by the service. The second model is fully trained and ready for use.

```
{
  "customizations": [
    {
      "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fa97",
      "created": "2016-06-01T18:42:25.324Z",
      "updated": "2020-01-19T11:12:02.296Z",
      "language": "en-US",
      "versions": [
        "en-US_BroadbandModel.v2018-07-31",
        "en-US_BroadbandModel.v2020-01-16"
      ],
      "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
      "name": "Example model one",
      "description": "Example custom acoustic model",
      "base_model_name": "en-US_BroadbandModel",
      "status": "pending",
      "progress": 0
    },
    {
      "customization_id": "8391f918-3b76-e109-763c-b7732faa3312",
      "created": "2017-12-01T18:51:37.291Z",
      "updated": "2017-12-02T19:21:06.825Z",
      "language": "en-US",
      "versions": [
        "en-US_BroadbandModel.v2017-11-15"
      ],
      "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
      "name": "Example model two",
      "description": "Example custom acoustic model two",
      "base_model_name": "en-US_BroadbandModel",
      "status": "available",
      "progress": 100
    }
  ]
}
```

List a specific custom acoustic model example

The following example returns information about the custom model that has the specified customization ID:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}"
```

The response duplicates the information from the previous example:

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fa97",
  "created": "2016-06-01T18:42:25.324Z",
  "updated": "2020-01-19T11:12:02.296Z",
  "language": "en-US",
  "versions": [
    "en-US_BroadbandModel.v2018-07-31",
    "en-US_BroadbandModel.v2020-01-16"
  ],
  "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
  "name": "Example model one",
  "description": "Example custom acoustic model",
  "base_model_name": "en-US_BroadbandModel",
  "status": "pending",
  "progress": 0
}
```

Resetting a custom acoustic model

Use the `POST /v1/acoustic_customizations/{customization_id}/reset` method to reset a custom acoustic model. Resetting a custom model removes all of the audio resources from the model, initializing the model to its state at creation. The method does not delete the model itself or metadata such as its name and language. However, when you reset a model, its audio resources are removed and must be re-created.

Reset a custom acoustic model example

The following example resets the custom acoustic model with the specified customization ID:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}/reset"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}/reset"
```

Deleting a custom acoustic model

Use the `DELETE /v1/acoustic_customizations/{customization_id}` method to delete a custom acoustic model that you no longer need. The method deletes all audio that is associated with the custom model and the model itself. Use this method with caution: A custom model and its data cannot be reclaimed after you delete the model.

Delete a custom acoustic model example

The following example deletes the custom acoustic model with the specified customization ID:

IBM Cloud

```
$ curl -X DELETE -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X DELETE \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}"
```

Managing audio resources

Note: Acoustic model customization is available only for previous-generation models. It is not available for next-generation and large speech models.

The customization interface includes the `POST /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method, which is used to add an audio resource to a custom acoustic model. For more information, see [Add audio to the custom acoustic model](). The interface also includes the following methods for listing and deleting audio resources for a custom acoustic model.

Listing audio resources for a custom acoustic model

The customization interface provides two methods for listing information about the audio resources of a custom acoustic model:

- The `GET /v1/acoustic_customizations/{customization_id}/audio` method lists information about all audio resources that have been added to a custom model.
- The `GET /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method lists information about a specified audio resource for a custom model.

Both methods return the `name` of the audio resource plus the following additional information:

- `duration` is the length in seconds of the audio. For an archive file, the figure represents the accumulated duration of all of the audio files that are contained in the archive.
- `details` identifies the type of the resource: `audio` or `archive`. (The type is `undetermined` if the service cannot validate the resource, possibly because the user mistakenly passed a file that does not contain audio.) For an audio file, the details include the `codec` and `frequency` of the audio. For an archive file, they include its `compression` type.

Additionally, listing all audio resources for a model returns the `total_minutes_of_audio` summed over all of the valid audio resources for the model. You can use this value to determine whether the custom model has enough or too much audio to begin training.

The methods also list the status of the audio data. The status is important for checking the service's analysis of audio files in response to a request to add them to a custom model:

- `ok` indicates that the service has successfully analyzed the audio data. The data can be used to train the custom model.
- `being_processed` indicates that the service is still analyzing the audio data. The service cannot accept requests to add new audio or to train the custom model until its analysis is complete.
- `invalid` indicates that the audio data is not valid for training the model (possibly because it has the wrong format or sampling rate, or because it is corrupted).

List all audio resources example

The following example lists all audio resources for the custom acoustic model with the specified customization ID:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}/audio"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}/audio"
```

The acoustic model has three audio resources. The service has successfully analyzed `audio1` and `audio2`; it is still analyzing `audio3`.

```
{
  "total_minutes_of_audio": 11.45,
  "audio": [
    {
      "duration": 131,
      "name": "audio1",
      "details": {
        "codec": "pcm_s16le",
        "type": "audio",
        "frequency": 22050
      }
      "status": "ok"
    },
    {
      "duration": 556,
      "name": "audio2",
      "details": {
        "type": "archive",
        "compression": "zip"
      },
      "status": "ok"
    },
    {
      "duration": 0,
      "name": "audio3",
      "details": {},
      "status": "being_processed"
    }
  ]
}
```

List an audio-type resource example

The following example returns information about the audio-type resource named `audio1`:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio1"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio1"
```

The resource is 131 seconds long and is encoded with the `pcm_s16le` codec. It was successfully added to the model.

```
{
  "duration": 131,
  "name": "audio1",
  "details": {
    "codec": "pcm_s16le",
    "type": "audio",
    "frequency": 22050
  }
  "status": "ok"
}
```

List an archive-type resource example

The following example returns information about the archive-type resource named `audio2`:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio2"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio2"
```

The resource is a `.zip` file that contains more than nine minutes of audio. It too was successfully added to the model. As the example shows, querying information about an archive-type resource also provides information about the files that it contains.

```
{
  "container": {
    "duration": 556,
    "name": "audio2",
    "details": {
      "type": "archive",
      "compression": "zip"
    },
    "status": "ok"
  },
  "audio": [
    {
      "duration": 121,
      "name": "audio-file1.wav",
      "details": {
        "codec": "pcm_s16le",
        "type": "audio",
        "frequency": 16000
      },
      "status": "ok"
    },
    {
      "duration": 133,
      "name": "audio-file2.wav",
      "details": {
        "codec": "pcm_s16le",
        "type": "audio",
        "frequency": 16000
      },
      "status": "ok"
    },
    . . .
  ]
}
```

Deleting an audio resource from a custom acoustic model

Use the `DELETE /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method to remove an existing audio resource from a custom acoustic model. When you delete an archive-type audio resource, the service removes the entire archive of files. The service does not allow deletion of individual files from an archive resource.

Removing an audio resource does not affect the custom model until you train the model on its updated data by using the `POST /v1/acoustic_customizations/{customization_id}/train` method. If you successfully trained the model on the resource, until you retrain the model, the existing audio data continues to be used for speech recognition.

Delete an audio resource example

The following method deletes the audio resource that is named `audio3` from the custom model with the specified customization ID:

IBM Cloud

```
$ curl -X DELETE -u "apikey:{apikey}" \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio3"
```

IBM Cloud Pak for Data

```
$ curl -X DELETE \
--header "Authorization: Bearer {token}" \
"{url}/v1/acoustic_customizations/{customization_id}/audio/audio3"
```

Grammars

Using grammars with custom language models

The IBM Watson® Speech to Text service supports the use of grammars with custom language models. You can add grammars to a custom language model and use them for speech recognition. Grammars restrict the set of phrases that the service can recognize from audio.

A grammar uses a formal language specification to define a set of production rules for transcribing strings. The rules specify how to form valid strings from the language's alphabet. When you apply a grammar to speech recognition, the service can return only one or more of the phrases that are generated by the grammar.

For example, when you need to recognize specific words or phrases, such as *yes* or *no*, individual letters or numbers, or a list of names, using grammars can be more effective than examining alternative words and transcripts. Moreover, by limiting the search space for valid strings, the service can deliver results faster and more accurately.

When you use a custom language model and a grammar for speech recognition, the service can return a valid phrase from the grammar or an empty result. If the result is not empty, the service includes a confidence score with the final transcript, as it does for all recognition requests. For grammars, the score indicates the likelihood that the response matched the grammar. False-positives are always possible, especially for simple grammars, so you must always consider the confidence of the service's results when evaluating its response.

Note: For more information about the languages and models that support grammars and their level of support (generally available or beta), see Language support for customization.

Supported grammar formats

The Speech to Text service supports grammars that are defined in the following standard formats:

- *Augmented Backus-Naur Form (ABNF)*, which uses a plain-text representation that is similar to traditional BNF grammar. The media type for this format is `application/srgs`.
- *XML Form*, which uses XML elements to represent the grammar. The media type for this format is `application/srgs+xml`.

Both grammar formats have the expressive power of a Context-Free Grammar (CFG). However, the service can decode only Type-3 Regular grammars in the Chomsky hierarchy. Such grammars represent finite state automata.

For general information about grammars, see the following Wikipedia pages:

- Speech Recognition Grammar Specification
- Augmented Backus-Naur form
- Chomsky hierarchy

The Speech Recognition Grammar Specification

The Speech to Text service supports grammars as defined by the W3C Speech Recognition Grammar Specification Version 1.0. The specification provides detailed information about the supported formats and about defining a grammar. For information about the supported media types, see Appendix G. Media Types and File Suffix of the specification.

The service does *not* currently support all features of the Speech Recognition Grammar Specification. Specifically, the service does not support the features described in the following sections of the specification:

- Section 1.4 Semantic Interpretation. IBM is working to support this feature in a future release of the service.
- Section 1.5 Embedded Grammars. IBM is working to support this feature in a future release of the service.
- Section 2.2.2 External Reference by URI. The service supports only local references, as described in Section 2.2.1 Local References. In other words, a grammar must be self-contained.
- Section 2.2.3 Special Rules.
- Section 2.2.4 Referencing N-gram Documents (Informative).
- Section 2.7 Language. The service does not support language switching. The service supports only one global language per grammar.

Note: Words in the grammar must be in UTF-8 encoding (ASCII is a subset of UTF-8). Using any other encoding can lead to issues when compiling the grammar or unexpected results in decoding. The service ignores an encoding specified in the header of the grammar.

Understanding grammars

The following examples introduce the IBM Watson® Speech to Text service's support for grammars. The examples create two simple ABNF grammars and show possible results when they are used for speech recognition. The examples illustrate the importance of examining the confidence score that the service includes with a transcript.

The examples provide only the results of speech recognition requests. For examples that show how to pass a grammar for speech recognition, see [Using a grammar for speech recognition](). The examples are also very basic. For examples of more complex grammars, see [Example grammars]().

Note: For more information about the languages and models that support grammars and their level of support (generally available or beta), see [Language support for customization]().

Single-phrase matches: The yesno grammar

The first example defines a very simple `yesno` grammar that accepts two valid single-word responses, `yes` and `no`. The grammar is useful in cases where the user must respond with only one of the two phrases.

```
#ABNF 1.0 ISO-8859-1;
language en-US;
mode voice;
root $yesno;

$yesno = yes | no ;
```

When you apply this grammar to a speech recognition request, the service can return a transcript with a score that indicates its confidence in the match. It can also return no result if the input clearly does not match one of the two phrases.

For instance, if the user replies `yes`, the service likely returns a response that is very much like the following result. The score in the `confidence` field indicates a perfectly reliable match.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "transcript": "yes"
        }
      ],
      "final": true
    }
  ]
}
```

But suppose, for example, the user replies `nope`. The service can return either a result with a very low confidence score or no result at all. An empty result is the clearest indication that the response does not match the grammar. An empty response is more likely to occur with complex grammars, where a valid response must match a specific multi-phrase sequence.

Multi-phrase matches: The names grammar

With a multi-phrase grammar, the user's response must be complete to be recognized. The user cannot omit a word or stop in the middle of the response. The absence of even a single word can cause the service to return an empty result.

Moreover, the service can return multiple transcripts if the user speaks phrases that are separated by sufficient silence to indicate that they are independent utterances. For example, consider the simple `names` grammar, which can match one of three multi-word names.

```
#ABNF 1.0 ISO-8859-1;
language en-US;
mode voice;
root $names;

$names = Yi Wen Tan | Yon See | Youngjoon Lee ;
```

Suppose the user speaks one of the names from the grammar's rules, `Yon See`. The service returns a response that indicates a very high level of confidence in the match.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.92,
          "transcript": "Yon See"
        }
      ],
      "final": true
    }
  ]
}
```

Now suppose that the user speaks two names separated by enough silence, at least 0.8 seconds, to indicate that they are separate utterances: `Yon See` [1.0 seconds of silence] `Yi Wen Tan`. In this case, the service sends two separate responses with a different confidence score for each result.

```
{
  "result_index": 0,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.92,
          "transcript": "Yon See"
        }
      ],
      "final": true
    }
  ]
},
{
  "result_index": 1,
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.83,
          "transcript": "Yi Wen Tan"
        }
      ],
      "final": true
    }
  ]
}
```

Now consider the case where the user instead says something like `Yon See` [1.0 seconds of silence] `Young Says He`. For the first phrase, `Yon See`, the service sends a positive match with a confidence score, similar to the previous examples. For the second phrase, `Young Says He`, the service can have one of two possible responses:

- It might send no response, indicating that the phrase does not match one of the grammar's rules.
- It might instead send a response with a low confidence score, indicating that the phrase is acoustically similar to one of the rules but is not a likely match.

Other factors and parameters can also effect grammar recognition:

- If the service encounters a pause that separates the phrases that are to be matched, the grammar fails to recognize the multi-phrase rule. You can use the `end_of_phrase_silence_time` parameter to increase the pause interval and avoid receiving multiple final results. For more information, see End of phrase silence time .
- If the `split_transcript_at_phrase_end` parameter is set to `true`, the service can split a transcript into multiple final results based on semantic features of the input. Such splits are unlikely to impact grammar recognition. But the use of a custom language model and a grammar can influence how the service interprets semantic features. For more information, see Split transcript at phrase end .

Confidence scores and empty results

For relatively simple grammars like those in the previous examples, the service can return a result with a low confidence score even when the response does not appear to match the grammar at all. This might seem surprising, but a response with a low confidence score can indicate the best match that the service could find for the given audio, even though the response is unlikely to match the grammar. If the response does not match the grammar, the confidence value of the results is typically low enough to indicate the unlikelihood that the grammar can generate the response.

Always consider the confidence score when evaluating whether a response satisfies a grammar. If you don't know what threshold to set for rejection of a result based on its confidence, use an initial value of 0.5. If you experience unexpectedly large rejection rates of correct results, decrease the confidence threshold; for instance, 0.1 can be a good option for some applications. If you experience large numbers of incorrect recognition results, increase the confidence threshold for your application.

In many cases, an empty result or a result with a very low confidence score is a valid response. It can indicate that the user did not understand the question or the available options. You can always recognize the user's audio response without the grammar, but you then run the risk of getting a result that is not valid for the grammar. Grammars provide more reliable results than n-grams, which always return a result for anything other than complete silence.

Knowing that the result must be one of the valid phrases defined by a grammar is a powerful feature that can simplify an application's response handling. Generally speaking, the service can return results with high confidence only for utterances that are generated by a grammar. For the `yesno` grammar in the first example to reliably recognize the phrase `nope` as a valid response, you must add that phrase to the grammar.

Adding a grammar to a custom language model

Before you can use a grammar for speech recognition, you must first use the customization interface to add the grammar to a custom language model. The steps to add a grammar to a custom language model parallel those used to add corpora or custom words:

1. Create a custom language model. You can create a new custom model or use an existing model.
2. Add a grammar to the custom language model. The service validates the grammar to ensure its correctness.
3. Validate the words from the grammar. You verify the correctness of the sounds-like pronunciations for any out-of-vocabulary (OOV) words that are recognized by the grammar.
4. Train the custom language model. The service prepares the custom model and grammar for use in speech recognition.
5. You can now use the custom model and grammar in speech recognition requests. For more information, see Using a grammar for speech recognition.

These steps are iterative. You can add grammars, as well as corpora and custom words, to a custom language model as often as needed. You must train the custom model on any new data resources (grammars, corpora, or custom words) that you add. When you use it for speech recognition, a custom model reflects the data on which it was last trained.

Note: For more information about the languages and models that support grammars and their level of support (generally available or beta), see Language support for customization.

Create a custom language model

To use a grammar with speech recognition, you must add it to a custom language model. You can use an existing custom language model, or you can create a new custom model by using the `POST /v1/customizations` method. For information about creating a new custom model, see Create a custom language model.

A custom language model can contain corpora and custom words as well as grammars. During speech recognition, you can use the custom model with or without its grammars. However, when you use a grammar, the service recognizes only words from the specified grammar.

Add a grammar to the custom language model

You use the `POST /v1/customizations/{customization_id}/grammars/{grammar_name}` method to add a grammar file to a custom model.

`customization_id` (*required* string)

Specify the customization ID of the custom model to which the grammar is to be added.

`grammar_name` (*required* string)

Specify a name for the grammar. Use a localized name that matches the language of the custom model and reflects the contents of the grammar.

- Include a maximum of 128 characters in the name.
- Do not use characters that need to be URL-encoded. For example, do not use spaces, slashes, backslashes, colons, ampersands, double quotes, plus signs, equals signs, questions marks, and so on in the name. (The service does not prevent the use of these characters. But because they must be URL-encoded wherever used, their use is strongly discouraged.)
- Do not use the name of a grammar or corpus that has already been added to the custom model.
- Do not use the name `user`, which is reserved by the service to denote custom words that are added or modified by the user.
- Do not use the name `base_lm` or `default_lm`. Both names are reserved for future use by the service.

`Content-Type` (*required* string)

Use the request header to specify the format of the grammar:

- `application/srgs` for an ABNF grammar
- `application/srgs+xml` for an XML grammar

Pass the grammar text file as the body of the request. The following example adds the grammar file named `confirm.abnf` to the custom model with the specified ID. The example names the grammar `confirm-abnf`.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: application/srgs" \
--data-binary @confirm.abnf \
"{url}/v1/customizations/{customization_id}/grammars/confirm-abnf"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: application/srgs" \
--data-binary @confirm.abnf \
"{url}/v1/customizations/{customization_id}/grammars/confirm-abnf"
```

The method also accepts an optional `allow_overwrite` query parameter that you can include to overwrite an existing grammar with the same name. Use the parameter if you need to update a grammar after you add it to a model.

The method is asynchronous. It can take a few seconds or minutes for the service to analyze the grammar, depending on the size of the grammar and the current load on the service. For information about checking the status of a grammar, see Monitoring the add grammar request.

You can add any number of grammars to a custom model by calling the method once for each grammar file. The addition of one grammar must be fully complete before you can add another.

Monitoring the add grammar request

The service returns a 201 response code if the grammar is valid. It then asynchronously processes the contents of the grammar and automatically extracts new words that the grammar can recognize. You cannot submit requests to add more data resources to a custom model, or to train the model, until the service's analysis of the grammar for the current request completes.

To determine the status of the analysis, use the `GET /v1/customizations/{customization_id}/grammars/{grammar_name}` method to poll the status of the grammar. The method accepts the ID of the custom model and the name of the grammar. The following example checks the status of the grammar named `confirm-abnf`.

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/grammars/confirm-abnf"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/grammars/confirm-abnf"
```

The response includes the status of the grammar:

```
{
  "name": "confirm-abnf",
  "out_of_vocabulary_words": 0,
  "status": "being_processed"
}
```

The `status` field has one of the following values:

- `analyzed` indicates that the service has successfully analyzed the grammar.

- `being_processed` indicates that the service is still analyzing the grammar.
- `undetermined` indicates that the service encountered an error while processing the grammar.

Use a loop to check the status of the grammar every 10 seconds until it becomes `analyzed`. For more information about checking the status of a grammar, see [Listing grammars for a custom language model]().

Handling add grammar failures

If its analysis of a grammar fails, the service sets the grammar's status to `undetermined` and includes an `error` field that describes the failure with the grammar's status. You can also use the `GET /v1/customizations/{customization_id}` method to check the status of the custom model. If addition of a grammar fails, the output includes an error message like the following:

```
{
  . . .
  "error": "{\"code\":500, \"code_description\":\"Internal Server Error\",
\". . . Cannot compile grammar . . .\"},
  . . .
}
```

The status of the custom model is not affected by the error, but the grammar cannot be used with the model. You can address the problem by correcting the grammar file and repeating the request to add it to the custom model. Set the `allow_overwrite` parameter to `true` to overwrite the version of the grammar file that failed.

You can also delete the grammar from the custom model for the time being, and then correct and add the grammar file again in the future. For more information about deleting a grammar, see [Deleting a grammar from a custom language model]().

Validate the words from the grammar

When you add a grammar file to a custom language model, the service parses the grammar to determine whether the grammar recognizes any words that are not already part of the service's base vocabulary. Such words are referred to as out-of-vocabulary (OOV) words. The service adds OOV words to the custom model's words resource. The purpose of the words resource is to define words that are not already present in the service's vocabulary.

Definitions in the words resource tell the service how to transcribe the OOV words. The information includes a `sounds_like` field that tells the service how the word is pronounced, a `display_as` field that tells the service how to display the word, and a `source` field that indicates how the word was added to the custom model. For more information about the words resource and OOV words, see [The words resource]().

After you add a grammar to a custom model, it is good practice to examine the OOV words in the model's words resource to verify their sounds-like pronunciations. Not all grammars have OOV words, but verifying the words resource is generally a good idea. To check the words of the custom model after you add a grammar, use the following methods:

- Use the `GET /v1/customizations/{customization_id}/words` method to list all of the words from a custom model. Pass the value `grammars` with the method's `word_type` parameter to list only words that were added from grammars.
- Use the `GET /v1/customizations/{customization_id}/words/{word_name}` method to view an individual word from a model.

Verify that the sounds-like pronunciations of the words are accurate and correct. Also look for typographical and other errors in the words themselves. For more information about validating and correcting the words in a custom model, see [Validating a words resource for previous-generation models]().

Train the custom language model

The final step before you can use a grammar with a custom language model is to train the model. You use the same process that you use for training a custom model on new corpora or new custom words. Training the model compiles the grammar for use during speech recognition. The service processes the grammar from its original text-based format to a binary runtime format for speech recognition. You cannot use the grammar until you train the model.

You use the `POST /v1/customizations/{customization_id}/train` method to train a custom model. You pass the method the customization ID of the model that you want to train, as in the following example.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/train"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/train"
```

The training method is asynchronous. Training typically takes only seconds or a few minutes to complete. But it can take longer depending on the size and complexity of the grammar and the current load on the service. For information about checking the status of a training operation, see Monitoring the training request.

Monitoring the training request

The service returns a 200 response code if the training process is successfully initiated. The service cannot accept subsequent training requests, or requests to add new grammars, corpora, or words to the custom model, until the current training request completes.

To determine the status of a training request, use the `GET /v1/customizations/{customization_id}` method to poll the model's status. The method accepts the customization ID of the model and returns information like the following about the model:

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96",
  "created": "2018-06-01T18:42:25.324Z",
  "language": "en-US",
  "dialect": "en-US",
  "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
  "name": "Example model",
  "description": "Example custom language model",
  "base_model_name": "en-US_BroadbandModel",
  "status": "training",
  "progress": 0
}
```

The `status` field has the value `training` while the model is being trained. Use a loop to check the status of the model every 10 seconds until it becomes `available`. For more information about monitoring a training request and other possible status values, see Monitoring the train model request.

Using a grammar for speech recognition

Once you create and train your custom language model with your grammar, you can use the grammar in speech recognition requests:

- Use the `language_customization_id` query parameter to specify the customization ID (GUID) of the custom language model for which the grammar is defined. A custom model can be used only with the base model for which it is created. If your custom model is based on a model other than the default, you must also specify that base model with the `model` query parameter. For more information, see Using the default model. You must issue the request with credentials for the instance of the service that owns the model.
- Use the `grammar_name` parameter to specify the name of the grammar. You can specify only a single grammar with a request.

When you use a grammar, the service recognizes only words from the specified grammar. The service does not use custom words that were added from corpora, that were added or modified individually, or that are recognized by other grammars.

Note: For more information about the languages and models that support grammars and their level of support (generally available or beta), see Language support for customization.

Examples of using a grammar with a custom language model

The following examples show the use of a grammar with a custom language model for each speech recognition interface:

- For the WebSocket interface, you first specify the customization ID with the `language_customization_id` parameter of the `/v1/recognize` method. You use this method to establish a WebSocket connection with the service.

  ```
  var access_token = {access_token};
  var wsURI = '{ws_url}/v1/recognize'
    + '?access_token=' + access_token
    + '&language_customization_id={customization_id}';
  var websocket = new WebSocket(wsURI);
  ```

  You then specify the name of the grammar with the `grammar_name` parameter in the JSON `start` message for the active connection. Passing this value with the `start` message allows you to change the grammar dynamically for each request that you send over the connection.

  ```
  function onOpen(evt) {
    var message = {
      action: 'start',
      content-type: 'audio/l16;rate=22050',
      grammar_name: '{grammar_name}'
    };
  ```

```
  websocket.send(JSON.stringify(message));
  websocket.send(blob);
}
```

- For the synchronous HTTP interface, pass both parameters with the `POST /v1/recognize` method.

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  --header "Content-Type: audio/flac" \
  --data-binary @audio-file.flac \
  "{url}/v1/recognize?language_customization_id={customization_id}&grammar_name={grammar_name}"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X POST  \
  --header "Authorization: Bearer {token}" \
  --header "Content-Type: audio/flac" \
  --data-binary @audio-file.flac \
  "{url}/v1/recognize?language_customization_id={customization_id}&grammar_name={grammar_name}"
  ```

- For the asynchronous HTTP interface, pass both parameters with the `POST /v1/recognitions` method.

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  --header "Content-Type: audio/flac" \
  --data-binary @audio-file.flac \
  "{url}/v1/recognitions?language_customization_id={customization_id}&grammar_name={grammar_name}"
  ```

  IBM Cloud Pak for Data

  ```
  $ curl -X POST \
  --header "Authorization: Bearer {token}" \
  --header "Content-Type: audio/flac" \
  --data-binary @audio-file.flac \
  "{url}/v1/recognitions?language_customization_id={customization_id}&grammar_name={grammar_name}"
  ```

Managing grammars

The customization interface includes the `POST /v1/customizations/{customization_id}/grammars/{grammar_name}` method for adding a grammar to a custom language model. For more information, see Add a grammar to the custom language model. The interface also includes the following methods for listing and deleting grammars for a custom language model.

Note: For more information about the languages and models that support grammars and their level of support (generally available or beta), see Language support for customization.

Listing grammars for a custom language model

The customization interface provides two methods for listing information about the grammars for a custom language model:

- The `GET /v1/customizations/{customization_id}/grammars` method lists information about all grammars for a custom model.
- The `GET /v1/customizations/{customization_id}/grammars/{grammar_name}` method lists information about a specified grammar for a custom model.

Both methods return the same information about a grammar. The information includes the `name` of the grammar and, *for custom models based on previous-generation models*, the number of `out-of_vocabulary_words` that are recognized by the grammar. The response also includes the `status` of the grammar, which is important for checking the service's analysis of the grammar when you add it to a custom model:

- `being_processed` indicates that the service is still processing the grammar in response to a `POST /v1/customizations/{customization_id}/grammars/{grammar_name}` request.

- `analyzed` indicates that the service has successfully processed the grammar and added it to the custom model.

- `undetermined` means that the service encountered an error while processing the grammar. The information that is returned for the grammar includes an error message that offers guidance for correcting the error.

For example, the grammar might be invalid, or you might have tried to add a grammar with the same name as an existing grammar. You can try to add the grammar again and include the `allow_overwrite` parameter with the request. You can also delete the grammar and then try adding it again.

List all grammars example

The following example lists information about all grammars that have been added to the custom model with the specified customization ID.

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/grammars"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/grammars"
```

Three grammars were successfully added to the custom model: `confirm-xml`, `confirm-abnf`, and `list-abnf`.

```
{
  "grammars": [
    {
      "out_of_vocabulary_words": 0,
      "name": "confirm-xml",
      "status": "analyzed"
    },
    {
      "out_of_vocabulary_words": 0,
      "name": "confirm-abnf",
      "status": "analyzed"
    },
    {
      "out_of_vocabulary_words": 8,
      "name": "list-abnf",
      "status": "analyzed"
    }
  ]
}
```

List a specific grammar example

The following example shows information about the specified grammar, `list-abnf`:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/grammars/list-abnf"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/grammars/list-abnf"
```

The grammar contains eight OOV words and is fully analyzed:

```
{
  "out_of_vocabulary_words": 8,
  "name": "list-abnf",
  "status": "analyzed"
}
```

Deleting a grammar from a custom language model

Use the `DELETE /v1/customizations/{customization_id}/grammars/{grammar_name}` method to remove an existing grammar from a custom language model. When it deletes the grammar, the service removes any OOV words that are associated with the grammar from the custom model's words resource unless

- The word was also added by another grammar or by a corpus.
- The word was modified in some way with the `POST /v1/customizations/{customization_id}/words` or `PUT /v1/customizations/{customization_id}/words/{word_name}` method.

Removing a grammar does not affect the custom model until you train the model on its updated data by using the `POST /v1/customizations/{customization_id}/train` method. If you successfully trained the model on the grammar, the grammar continues to be available for speech recognition until you retrain the model.

Delete a grammar example

The following example deletes the grammar named `list-abnf` from the custom model with the specified customization ID.

IBM Cloud

```
$ curl -X DELETE -u "apikey:{apikey}" \
"{url}/v1/customizations/{customization_id}/grammars/list-abnf"
```

IBM Cloud Pak for Data

```
$ curl -X DELETE \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations/{customization_id}/grammars/list-abnf"
```

Example grammars

The following examples describe more complex uses for grammars with speech recognition. Most examples are provided in both ABNF (`application/srgs`) and XML (`application/srgs+xml`) formats. Use these examples to help you get started with grammars.

Tip: In a production rule for a grammar, including a `\` (backslash) is a syntax error in ABNF and is interpreted as a literal value in XML.

Note: For more information about the languages and models that support grammars and their level of support (generally available or beta), see Language support for customization.

Confirmation grammars

Confirmation grammars are useful for applications that expect a one-word answer in response to a question. The following grammars define a list of possible yes and no responses.

- **ABNF** - confirm.abnf
- **XML** - confirm.xml

List grammars

List grammars are useful for applications that expect the user to select an option from a predefined set of strings. The set is sometimes called a flat list. It typically consists of a list of terminal elements and does not include a complicated rule structure.

- The following grammars define a list of phrases for digits:
  - **ABNF** - list-numbers.abnf
  - **XML** - list-numbers.xml
- The following grammar defines a list of valid names from which the user can choose, perhaps to select an individual from a telephone directory:
  - **ABNF** - list-names.abnf
  - **XML** - Not available

Vehicle identification number grammars

Vehicle identification numbers (VINs) use a fixed alphanumeric format, as do credit card numbers, telephone numbers, and US social security numbers. A great advantage of grammars over classical n-grams is that grammars are very effective for specifying such formats.

The VIN format is well standardized and has a fixed number of characters. Classic n-grams perform poorly for these types of tasks. Unlike grammars, they cannot ensure that the required number of characters is provided.

The following grammars recognize Honda VIN codes. They are more complex than the previous examples but demonstrate the power of grammars nicely.

- **ABNF** - vins.abnf
- **XML** - vins.xml

For more information about the VIN format, see Vehicle identification number.

Grammars with optional elements

By making certain elements of a response optional, you can make grammars more flexible by anticipating how users might respond. The following grammar adds square brackets around the element `$optionalphrase` to make it optional. The user can speak one of a few additional phrases before stating the social security number. For instance, the user can say "my social is xxx xx xxxx" or just "xxx xx xxxx".

- **ABNF** - optional.abnf
- **XML** - Not available

For more information about optional expansions in grammars, see Section 2.5 Repeats of the Speech Recognition Grammar Specification.

Disambiguation grammars

Other possible example grammars include situations in which the confidence of the recognized response is not very high but the application needs to be certain of the user's response. In this case, the application can dynamically create a grammar that includes the most likely responses and ask the user to repeat the answer. For example, the application can create a grammar that includes the two most likely options and ask the user to select one of the two: `Did you mean 'option 1' or 'option 9'?`

Disambiguation grammars are programmatically generated. No examples are provided.

Custom model upgrading

Upgrading custom models

To improve the quality of speech recognition, the IBM Watson® Speech to Text service occasionally updates base previous- and next-generation models. Because base models for different languages are independent of each other, as are different models for the same language, updates to individual base models do not affect other models.

When a base model is updated, you might need to upgrade any custom models that are built on that base model to take advantage of the improvements. An update to a base model that requires an upgrade produces a new version of the base model (for example, `en-US_BroadbandModel.v2020-01-16`). An update that does not require an upgrade does not produce a new version.

- *For previous-generation models,* all updates to a base model produce a new version of the model. When a new version of a base model is released, you must upgrade any custom language and custom acoustic models that are built on the updated base model to take advantage of the improvements.
- *For next-generation models,* most updates do not produce a new version of the model. These updates do not require that you upgrade your custom models. Some updates, however, do generate a new version of a base model. You must upgrade any custom language models that are built on the updated base model to take advantage of the improvements.

  The service released new language model customization technology on 27 February 2023. How you upgrade to the new technology is different from traditional custom model upgrades. For more information about the new technology and about upgrading, see

  - Improved language model customization for next-generation models
  - Upgrading a custom language model based on an improved next-generation model

If an upgrade is required, your custom models continue to use the older version of the base model until you complete the upgrade. Upgrade to the latest version of an updated base model as soon as possible. The sooner you upgrade a custom model, the sooner you can experience the improved speech recognition of the new model.

Once you upgrade a custom model, the service uses the latest version of the custom model by default when you specify that model with a speech recognition request. But you can still direct the service to use the older version of the model. Note that older versions of base models can be removed at a future time. To encourage upgrading, the service returns a warning message with the results for recognition requests that use custom models that are based on older base models.

All updates to base models and whether they require upgrades are announced in the release notes:

- [Release notes for Speech to Text for IBM Cloud](#)
- [Release notes for Speech to Text for IBM Cloud Pak for Data](#)

As with all customization operations, you must use credentials for the instance of the service that owns a custom model to upgrade it.

How upgrading works

When a new version of a base model is first released, existing custom models are still based on the older version of the base model. Until a custom model is upgraded, all operations on that custom model, such as adding data to or training the model, affect the existing version of the custom model. Similarly, all recognition requests that specify the custom model use the existing version of the custom model.

Upgrading results in two versions of a custom model, one based on the older version of the base model and one based on the latest version of the base model. Once a custom model is upgraded, all operations on that custom model affect the newer, upgraded version of the custom model. It is no longer possible to add data to or to train the older version of the model. You cannot undo an upgrade operation.

When you upgrade a custom model, you do not need to upgrade its individual resources. For a custom language model, the service automatically upgrades any corpora, grammars, and words that are defined for the model. Likewise, when you upgrade a custom acoustic model, the service automatically upgrades its audio resources. You do not need to upgrade a custom language or custom acoustic model that contains no resources.

By default, the service uses the latest version of a custom model for speech recognition requests. However, you can continue to use the older version of a custom model for speech recognition. For more information, see [Using upgraded custom models for speech recognition](#).

Upgrading a custom language model

Follow these steps to upgrade a custom language model:

1. Ensure that the custom language model is in either the `ready` or the `available` state. You can check a model's status by using the `GET /v1/customizations/{customization_id}` method. If the model's state is `ready`, upgrade the model before training it on its latest data.
2. Upgrade the custom language model by using the `POST /v1/customizations/{customization_id}/upgrade_model` method:

   IBM Cloud

   ```
   $ curl -X POST -u "apikey:{apikey}" \
   "{url}/v1/customizations/{customization_id}/upgrade_model"
   ```

   IBM Cloud Pak for Data

   ```
   $ curl -X POST \
   --header "Authorization: Bearer {token}" \
   "{url}/v1/customizations/{customization_id}/upgrade_model"
   ```

   The upgrade method is asynchronous. It can take on the order of minutes to complete depending on the number of words in the model's words resource and the current load on the service.

The service returns a 200 response code if the upgrade process is successfully initiated. You can monitor the status of the upgrade by using the `GET /v1/customizations/{customization_id}` method to poll the model's status. Use a loop to check the status every 10 seconds.

While it is being upgraded, the custom model has the status `upgrading`. When the upgrade is complete, the model resumes the status that it had before upgrade (`ready` or `available`). Checking the status of an upgrade operation is identical to checking the status of a training operation. For more information, see [Monitoring the train model request](#).

The service cannot accept requests to modify the model in any way until the upgrade request completes. However, you can continue to issue recognition requests with the existing version of the model during the upgrade.

Upgrading a custom language model based on an improved next-generation model

For custom language models that are based on improved next-generation models, upgrade is performed automatically when you retrain the custom model. And once a custom model is based on an improved next-generation model, the service continues to perform any necessary upgrades when the custom model is retrained.

Note: You cannot use the `POST /v1/customizations/{customization_id}/upgrade_model` method to upgrade a custom model to an improved next-generation base model.

To upgrade a custom model that is based on an improved next-generation model, perform one of the following procedures:

- Change the custom model and then train it.

    1. Change your custom model by adding or modifying a custom word, corpus, or grammar that the model contains. Any change that you make moves the model to the `ready` state.
    2. Use the `POST /v1/customizations/{customization_id}/train` method to retrain the model. Retraining upgrades the custom model and moves it to the `available` state.

- IBM Cloud Train the custom model by using the `force` query parameter.

  If you include the parameter `force=true` with the `POST /v1/customizations/{customization_id}/train` request, you do not need to change the model's custom words, corpora, or grammars to enable upgrading. The parameter forces the training, and thus the upgrade, of the custom model regardless of whether it is in the `ready` or `available` state. The following example shows the use of the `force` parameter:

  IBM Cloud

  ```
  $ curl -X POST -u "apikey:{apikey}" \
  "{url}/v1/customizations/{customization_id}/train?force=true"
  ```

For more information about training a custom model and about monitoring a training request, see [Train the custom language model](#).

Tip: To identify models that use improved language model customization, look for an *(improved)* date in the *Language model customization* column of table 2 in [Customization support for next-generation models](#). Make sure to check for a date for the version of the service that you use, *IBM Cloud* or *IBM Cloud Pak for Data* .

Upgrading a custom acoustic model

Follow these steps to upgrade a custom acoustic model. If the custom acoustic model was trained with a custom language model, you must perform two extra upgrade steps where indicated.

1. *If the custom acoustic model was trained with a custom language model,* you must first upgrade the custom language model to the latest version of the base model. For more information, see [Upgrading a custom language model](#).
2. Ensure that the custom acoustic model is in either the `ready` or the `available` state. You can check a model's status by using the `GET /v1/acoustic_customizations/{customization_id}` method. If the model's state is `ready`, upgrade the model before training it on its latest data.
3. Upgrade the custom acoustic model by using the `POST /v1/acoustic_customizations/{customization_id}/upgrade_model` method:

   IBM Cloud

   ```
   $ curl -X POST -u "apikey:{apikey}" \
   "{url}/v1/acoustic_customizations/{customization_id}/upgrade_model"
   ```

   IBM Cloud Pak for Data

   ```
   $ curl -X POST \
   --header "Authorization: Bearer {token}" \
   "{url}/v1/acoustic_customizations/{customization_id}/upgrade_model"
   ```

   The upgrade method is asynchronous. It can take on the order of minutes or hours to complete, depending on the amount of audio data that the model contains and the current load on the service. As with training, upgrading generally takes as long as the cumulative amount of audio data that the model contains.
4. *If the custom acoustic model was trained with a custom language model,* upgrade the custom acoustic model again, this time with the previously upgraded custom language model. Use the `custom_language_model_id` query parameter to specify the customization ID of the custom language model.

   IBM Cloud

   ```
   $ curl -X POST -u "apikey:{apikey}" \
   "{url}/v1/acoustic_customizations/{customization_id}/upgrade_model?custom_language_model_id={custom_language_model_id}"
   ```

   IBM Cloud Pak for Data

   ```
   $ curl -X POST \
   --header "Authorization: Bearer {token}" \
   "{url}/v1/acoustic_customizations/{customization_id}/upgrade_model?custom_language_model_id={custom_language_model_id}"
   ```

Once again, the upgrade method is asynchronous, and upgrading generally takes as long as the cumulative amount of audio that the model contains.

Note: The request to upgrade the acoustic model with the language model might fail with a 400 response code and the message `No input data modified since last training`. If this error occurs, add the boolean `force` query parameter to the request and set the parameter to `true`. Use the parameter only to force an upgrade of a custom acoustic model in this particular situation.

The service returns a 200 response code if the upgrade process is successfully initiated. You can monitor the status of the upgrade by using the `GET /v1/acoustic_customizations/{customization_id}` method to poll the model's status. Use a loop to check the status periodically (for example, once a minute).

While it is being upgraded, the custom model has the status `upgrading`. When the upgrade is complete, the model resumes the status that it had before upgrade (`ready` or `available`). Checking the status of an upgrade operation is identical to checking the status of a training operation. For more information, see Monitoring the train model request.

The service cannot accept requests to modify the model in any way until the upgrade request completes. However, you can continue to issue recognition requests with the existing version of the model during the upgrade.

Upgrade failures

The upgrade of a custom model fails to start if the service is handling another request for the model, such as a request to add data or a training request. The upgrade request also fails to start in the following cases:

- The custom model is in a state other than `ready` or `available`.
- The custom model contains no data (corpora, custom words, grammars, or audio resources).
- For a custom acoustic model that was trained with a custom language model, the custom models are based on different versions of the base model. You must upgrade the custom language model before upgrading the custom acoustic model. For more information, see Upgrading a custom acoustic model.

Using upgraded custom models for speech recognition

Once you upgrade a custom model, the service uses the latest version of the custom model by default when you specify that model with a speech recognition request. However, you can still direct the service to use the older version of the model. You need to list information about the custom model to determine the available versions.

Listing version information for a custom model

Listing information about a custom model is the only way to see the available versions of a base model. To see the versions of the base model for which a custom model is available, use the following methods:

- To list information about a custom language model, use the `GET /v1/customizations/{customization_id}` method. For more information, see Listing custom language models.
- To list information about a custom acoustic model, use the `GET /v1/acoustic_customizations/{customization_id}` method. For more information, see Listing custom acoustic models.

In both cases, the output includes a `versions` field that shows information about the base models that are available for the custom model. If the custom model has not been upgraded or only a single version of its base model exists, the `versions` field shows a single version.

Example of listing version information

The following example request shows information for an upgraded custom language model:

IBM Cloud

```
$ curl -X GET -u "apikey:{apikey}" \
"{url}/v1/customizations?language=en-US"
```

IBM Cloud Pak for Data

```
$ curl -X GET \
--header "Authorization: Bearer {token}" \
"{url}/v1/customizations?language=en-US"
```

The `versions` field indicates that the custom model is available for two versions of the base model: the older version, `en-US_BroadbandModel.v2018-07-31`, and the newer version, `en-US_BroadbandModel.v2020-01-16`.

```
{
  "customization_id": "74f4807e-b5ff-4866-824e-6bba1a84fe96",
  "created": "2016-06-01T14:21:26.894Z",
  "updated": "2020-01-18T18:42:25.324Z",
  "language": "en-US",
  "dialect": "en-US",
  "versions": [
    "en-US_BroadbandModel.v2018-07-31",
    "en-US_BroadbandModel.v2020-01-16"
  ],
  "owner": "297cfd08-330a-22ba-93ce-1a73f454dd98",
  "name": "Example model",
  "description": "Example custom language model",
  "base_model_name": "en-US_BroadbandModel",
  "status": "available",
  "progress": 100
}
```

Making speech recognition requests with upgraded custom models

By default, the service uses the latest version of a custom model that you pass with a speech recognition request. You issue requests with upgraded custom models as you do with any model. For more information, see

- Using a custom language model for speech recognition
- Using a custom acoustic model for speech recognition
- Using a grammar for speech recognition

However, you can also continue to make speech recognition requests with the older version of the custom model by using the `base_model_version` query parameter of a request. You specify the version of the base model that is identified in the `versions` field of the listing output, and the service uses that base model for speech recognition.

You can use the parameter to test the performance and accuracy of a custom model against both the old and new versions of its base model. If you find that an upgraded model's performance is lacking in some way (for instance, certain words are no longer recognized), you can continue to use the older version with recognition requests while you address the inadequacies.

The `base_model_version` parameter is intended primarily for use with custom models that are updated for a new base model, but it can be used without a custom model. The version of a base model that is used for a request depends both on whether you pass the `base_model_version` parameter and on whether you specify a custom model (language, acoustic, or both) with the request:

- *If you specify a custom model with the request:*
  - Omit the `base_model_version` parameter to use the latest version of the base model to which the custom model is upgraded. For example, if the custom model is upgraded to the latest version of the base model, the service uses the latest versions of the base and custom models.
  - Specify the `base_model_version` parameter to use the specified versions of both the base and custom models.
- *If you do not specify a custom model with the request:*
  - Omit the `base_model_version` parameter to use the latest version of the base model.
  - Specify the `base_model_version` parameter to use a specific version of the base model.

To encourage upgrading, the service returns a warning message with speech recognition results that use custom models that are based on older base models.

Note: Because the `base_model_version` parameter is intended primarily for use with custom models, listing information about a custom model is the only way to see the available versions of a base model.

Considerations for using custom acoustic and custom language models together

In addition to the behavior described in the previous topic, the service considers the upgrade status of the models that are passed with a speech recognition request. If you pass both custom language and custom acoustic models with a request, the service applies the following guidelines when deciding on the base model version to use with the request:

- If both custom models are based on the older base model, the service uses the old base model for recognition.
- If both custom models are based on the newer base model, the service uses the new base model for recognition.
- If only one of the two custom models is upgraded to the newer base model, the service uses the old base model for recognition. The old base model is the version that the two custom models have in common.

Both custom models must be owned by the credentials that are passed with the request, both must be based on the same base model (for example, `en-US_BroadbandModel`), and both must be in the `available` state. For more information about using both types of custom models with a recognition request, see [Using custom language and custom acoustic models for speech recognition](#).

Example of using an upgraded custom model

The earlier example listed the following available versions for a custom language model:

```
"versions": [
  "en-US_BroadbandModel.v2018-07-31",
  "en-US_BroadbandModel.v2020-01-16"
],
```

The following synchronous HTTP request specifies that the older version of the base model is to be used. Thus, the older version of the specified custom language model is also used.

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?model=en-US_BroadbandModel&base_model_version=en-US_BroadbandModel.v2018-07-31&language_customization_id=
{customization_id}"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "Content-Type: audio/flac" \
--data-binary @{path}audio-file.flac \
"{url}/v1/recognize?model=en-US_BroadbandModel&base_model_version=en-US_BroadbandModel.v2018-07-31&language_customization_id=
{customization_id}"
```

For an asynchronous HTTP request, you specify the parameters when you create the asynchronous job. For a WebSocket request, you pass the parameters when you establish a connection.

Security topics

Information security

IBM® is committed to providing our clients and partners with innovative data privacy, security, and governance solutions.

Important: Clients are responsible for ensuring their own compliance with various laws and regulations, including the European Union General Data Protection Regulation. Clients are solely responsible for obtaining advice of competent legal counsel as to the identification and interpretation of any relevant laws and regulations that might affect the clients' business and any actions the clients might need to take to comply with such laws and regulations.

The products, services, and other capabilities described herein are not suitable for all client situations and might have restricted availability. IBM does not provide legal, accounting or auditing advice or represent or warrant that its services or products will ensure that clients are in compliance with any law or regulation.

If you need to request GDPR support for IBM Cloud® Watson resources that are created

- In the European Union (EU), see Requesting support for IBM Cloud Watson resources created in the European Union.
- Outside of the EU, see Requesting support for resources outside the European Union.

European Union General Data Protection Regulation (GDPR)

IBM is committed to providing our clients and partners with innovative data privacy, security and governance solutions to assist them on their journey to GDPR compliance.

Learn more about IBM's own GDPR readiness journey and our GDPR capabilities and offerings to support your compliance journey here.

Health Insurance Portability and Accountability Act (HIPAA)

IBM Cloud

US Health Insurance Portability and Accountability Act (HIPAA) support is available for Premium plans that are hosted in the Washington, DC, (`us-east`) and Dallas (`us-south`) locations. For more information, see Enabling EU and HIPAA supported settings.

Do not include personal health information (PHI) in data that is to be added to custom models. Specifically, be sure to remove any PHI from data that you use for custom language models or custom acoustic models.

Labeling and deleting data in the Speech to Text service

The IBM Watson® Speech to Text service enables you to delete all data that is associated with recognition requests, custom language models, and custom acoustic models. To delete data, you must do the following:

1. Use the `X-Watson-Metadata` header to associate a customer ID with data that is passed by a request to the service; see Specifying a customer ID.
2. Use the `DELETE /v1/user_data` method to delete all data that is associated with a specified customer ID; see Deleting data.

By default, no customer ID is associated with data.

Note: Experimental and beta features are not intended for use with a production environment and therefore are not guaranteed to function as expected when labeling and deleting data. Experimental and beta features should not be used when implementing a solution that requires the labeling and deletion of data.

Specifying a customer ID

To associate a customer ID with data, include the `X-Watson-Metadata` header with the request that passes the information. You pass the string `customer_id={id}` as the argument of the header.

A customer ID can include any characters except for the `;` (semicolon) and `=` (equals sign). Specify a random or generic string for the customer ID; do not specify a personally identifiable string, such as an email address or Twitter ID. You can specify different customer IDs with different requests. A customer ID that you specify is associated with the instance of the service whose credentials are used with the request; only credentials for that instance of the service can delete data associated with the ID.

Supported methods

You can use the `X-Watson-Metadata` header with the following methods:

- With WebSocket requests:
  - `/v1/recognize`

  You specify the customer ID with the `x-watson-metadata` query parameter of the request to open the connection. You must URL-encode the argument to the query parameter, for example, `customer_id%3dmy_customer_ID`. The customer ID is associated with all data that is passed with recognition requests sent over the connection.

- With synchronous HTTP requests:
  - `POST /v1/recognize`

  The customer ID is associated with the data that is sent with the individual request.

- With asynchronous HTTP requests:
  - `POST /v1/register_callback`
  - `POST /v1/recognitions`

  The customer ID is associated with the allowlisted callback URL or with the data that is sent with the individual recognition request.

- With requests to add corpora, custom words, or grammars to custom language models:
  - `POST /v1/customizations/{customization_id}/corpora/{corpus_name}`
  - `POST /v1/customizations/{customization_id}/words`
  - `PUT /v1/customizations/{customization_id}/words/{word_name}`
  - `POST /v1/customizations/{customization_id}/grammars/{grammar_name}`

  The customer ID is associated with the corpora, custom words, or grammars that are added or updated by the request.

- With requests to add audio resources to custom acoustic models:
  - `POST /v1/acoustic_customizations/{customization_id}/audio/{audio_name}`

  The customer ID is associated with the audio resource that is added or updated by the request.

Specify a customer ID example

The following example associates the customer ID `my_customer_ID` with the data passed with a `POST /v1/recognize` request:

IBM Cloud

```
$ curl -X POST -u "apikey:{apikey}" \
--header "X-Watson-Metadata: customer_id=my_customer_ID" \
--header "Content-Type: audio/wav" \
--data-binary @audio.wav \
"{url}/v1/recognize"
```

IBM Cloud Pak for Data

```
$ curl -X POST \
--header "Authorization: Bearer {token}" \
--header "X-Watson-Metadata: customer_id=my_customer_ID" \
--header "Content-Type: audio/wav" \
--data-binary @audio.wav \
"{url}/v1/recognize"
```

Deleting customer data

To delete all data that is associated with a customer ID, use the `DELETE /v1/user_data` method. You pass the string `customer_id={id}` as a query parameter with the request.

The `/v1/user_data` method deletes all data that is associated with the specified customer ID, regardless of the method by which the information was added. The method has no effect if no data is associated with the customer ID. You must issue the request with credentials for the same instance of the service that was used to associate the customer ID with the data.

Delete customer data example

The following example deletes all data for the customer ID `my_customer_ID`:

IBM Cloud

```
$ curl -X DELETE -u "apikey:{apikey}" \
"{url}/v1/user_data?customer_id=my_customer_ID"
```

IBM Cloud Pak for Data

```
$ curl -X DELETE \
--header "Authorization: Bearer {token}" \
"{url}/v1/user_data?customer_id=my_customer_ID"
```

Deletion of all data for a Speech to Text service instance

IBM Cloud

If you delete an instance of the Speech to Text service from the IBM Cloud console, all data associated with that service instance is automatically deleted. This includes all custom language models, corpora, grammars, and words; all custom acoustic models and audio resources; all registered endpoints for the asynchronous HTTP interface; and all data related to speech recognition requests.

This data is purged automatically and regardless of whether a customer ID is associated with the data. Once you delete a service instance, you can no longer restore any of the deleted data.

Protecting sensitive information in your Watson service

IBM Cloud

You can add a higher level of encryption protection and control to your data at rest (when it is stored) and data in motion (when it is transported) by enabling integration with IBM® Key Protect for IBM Cloud®.

The data that you store in IBM Cloud is encrypted at rest by using a randomly generated key. If you need to control the encryption keys, you can integrate Key Protect. This process is commonly referred to as *Bring your own keys* (BYOK). With Key Protect you can create, import, and manage encryption keys. You can assign access policies to the keys, assign users or service IDs to the keys, or give the key access only to a specific Watson service. The first 20 keys are free.

Tip: Data encryption of Watson services requires a new Premium plan instance. You cannot encrypt an existing service instance or instances not in the Premium plan. Not all Watson services support Premium plans. For more information about Premium plans, contact a Watson representative

About customer-managed encryption

Watson uses *envelope encryption* to implement customer-managed keys. Envelope encryption describes encrypting one encryption key with another encryption key. The key used to encrypt the actual data is known as a *data encryption key (DEK)*. The DEK itself is never stored but is wrapped by a second key that is known as the key encryption key (KEK) to create a wrapped DEK. To decrypt data, the wrapped DEK is unwrapped to get the DEK. This process is possible only by accessing the KEK, which in this case is your root key that is stored in Key Protect.

Key Protect keys are secured by FIPS 140-2 Level 3 certified cloud-based *hardware security modules (HSMs)*.

Enabling customer-managed keys with Watson

Important: Some Watson services have additional details about how to work with Key Protect. For more information, see the docs for the service.

Integrating Key Protect with Watson Premium services involves these steps in the IBM Cloud console.

1. Create an instance of Key Protect.
2. Add a root key to the Key Protect instance.
3. Grant Key Protect access to all instances of your Watson service.
4. Encrypt the Watson service data

Step 1. Create the Key Protect instance

Create an instance of Key Protect to hold your root keys.

1. Go to the Key Protect page in the IBM Cloud catalog **Security and Identity** category.
2. Select a region. Make sure to create the instance in the same location as the Watson services you want to encrypt.

3. Name the service and click **Create**.

Step 2. Add a key

You use Key Protect to generate a key or import your own key.

Create a root key

When you use Key Protect to create a key, the service generates cryptographic key material that is rooted in cloud-based HSMs.

After you create an instance of the service, add a root key.

1. If you're not already on the details page, click the name of the Key Protect instance in your resource list.
2. Add a Key Protect *root key*:
    a. Click **Manage** from the left navigation pane of the service details and click **Add key**.
    b. Select **Create a key** and the **Root key** type.
    c. Give the key a name that you can recognize and click **Create key**. Make sure that the key name does not contain personal information, such as your name or location.

Import a key

If you need to generate keys with your own solution, you can use Key Protect to secure the keys.

After you create an instance of the service, add a root key.

1. Click the name of the Key Protect instance in your resource list.
2. Import a root key:
    a. Click **Manage** from the left navigation pane of the service details and click **Add key**.
    b. Select **Import your own key** the **Root key** type.
    c. In **Key material**, specify the base64 encoded key material, such as an existing key-wrapping key.
        - The key must be 128, 192, or 256 bits.
        - The bytes of data, for example 32 bytes for 256 bits, must be encoded by using base64 encoding.
    d. Give the key a name that you can recognize and click **Import key**. Make sure that the key name does not contain personal information, such as your name or location.

Step 3. Grant service access to Key Protect

After you add a root key to Key Protect, you use Cloud Identity and Access Management (IAM) to authorize access between the Watson service and Key Protect.

You must be the account owner or administrator on the Key Protect service instance and at least the viewer role on all instances of the Watson services.

1. Go to the IAM Authorizations page. (From the IBM Cloud console menu bar, select **Manage** > **Access IAM**, and then click **Authorizations**.)
2. Click **Create**.
3. Select the Watson service as the **Source service**. Leave **All resources** selected to scope the access.
4. Select Key Protect as the **Target service**.
5. Select **Resources based on selected attributes** , and then select **Instance ID** for the target service attribute. One of the following criteria *must* be true to authorize:
    - The IAM Policy authorizes a resource group that contains the Key Protect instance to target all instances of the dependent service.
    - The IAM Policy authorizes a specific instance of Key Protect to target all instances of the dependent service.
    - The IAM Policy authorizes a specific instance of Key Protect to target a resource group that the new instance of the dependent service would be created in.
    - The IAM Policy authorizes a resource group that contains the Key Protect instance to target a resource group that the new instance of the dependent service would be created in.

    ⚠ **Important:** Select the Key Protect service instance that you created earlier.

6. Authorize dependent services by selecting **Enable authorization to be delegated**. Delegation allows the Key Protect instance to propagate its authorizations to this service.
7. Make sure that the **Reader** role is enabled. Reader permissions allow the Watson service to see the root keys in the Key Protect instance.
8. Click **Authorize** and confirm delegated authorization.

Step 4. Encrypt the Watson service data

After you grant the Watson service the authorization to use your keys, you create another instance on your Premium plan and supply the key name or CRN. The service uses your encryption key to encrypt your data.

You must have the administrator or editor role on the Watson service.

1. Go to the **AI / Machine Learning** category page.
2. Select the Watson service that you authorized in Step 3. Grant service access
3. Select a region. Make sure to create the instance in the same location as the Key Protect service that you created in Step 1.
4. Select the Premium plan. This feature is supported on the Premium plan only.
5. Encrypt the service data with Key Protect:
   a. Select **Yes** to "Encrypt service with Key Protect."
   b. Select the Key Protect instance that you authorized earlier.
   c. Select the root key that you authorized earlier.

   Tip: If you see a drop-down menu flash and disappear, check that you have correctly followed sub-step 5 of Step 3. Grant service access

6. Click **Create**.

The data that is associated with this Watson service instance is now encrypted.

Working with customer-managed keys

After you enable a customer-managed key, the service operates normally, and you can manage access to the data with Key Protect.

Temporarily prevent access to your data

To temporarily prevent access, remove all authorizations between the Watson service and Key Protect:

- Remove the authorization that you created in Step 3 from the IAM Authorizations page.
- Remove all other authorizations between other services connected with the Watson service and Key Protect.
  1. Find the Cloud Resource Name identifier (CRN) for the Watson service that you want to remove access to. You find the CRN by clicking the service instance row in your Resource list. The CRN is displayed in the details pane.
  2. Back on the Authorizations page, search for that CRN in the list.
  3. Remove every authorization with the CRN of the Watson service as the source and Key Protect as the target.

  These other authorizations might exist because the Watson service can create delegated policies from the authorization that you created.

The Watson instance can no longer access the data because no authorizations exist to access the key. For more information, search for Removing an authorization in "Using authorizations to grant access between services".

Restore temporary access

To restore access after you temporarily remove it, follow these steps:

1. Re-create the authorization between your Watson service and Key Protect instance as shown in Step 3. Grant service access.
2. Tell your IBM Client Success Manager (CSM) that you want to restore access through Key Protect.

After you re-create the authorization and your CSM confirms Watson that any delegated authorizations are reconnected, the Watson instance starts accepting connections again.

Permanently prevent access to your data

To delete your data securely, delete both the encrypted Watson service instance and the Key Protect key.

⚠ **Important:** When you delete a data encryption key, it is not recoverable and you cannot decrypt the key. You prevent further access to the data, but you also cannot recover the data.

For more information about deleting keys, see [Deleting keys](#) in the Key Protect docs.

Rotating keys

Key rotation is an important part of mitigating the risk of a data breach. Periodically changing keys reduces the potential data loss if the key is lost or compromised. For more information, see [Setting a rotation policy](#) in the Key Protect docs.

Identify which key is encrypting your service instance

To see which Key Protect instance is used to encrypt the data, use the link in the service details.

1. Go to your [resource list](#).
2. Click the name of the Watson service instance.
3. Click the **Manage** tab on the left pane.
4. Find "Encrypted with a Key Protect root key". Click **View KeyProtect Instance** to see details of the keys in that Key Protect instance.

Next steps

- Read more about [Bringing your encryption keys to the cloud](#) in the Key Protect docs.
- See which [Activity Tracker events](#) are recorded with Key Protect.

Public and private network endpoints

IBM Cloud

IBM Cloud® supports both public and private network endpoints for certain plans. Connections to private network endpoints do not require public internet access.

Private network endpoints support routing services over the IBM Cloud private network instead of the public network. A private network endpoint provides a unique IP address that is accessible to you without a VPN connection.

Enabling your account

⚠ **Important:** Private network endpoints are supported for paid plans. Check the plan information for your service to learn about the plans that support private network endpoints.

Your account must be configured before you can use private endpoints. To use private network endpoints, the following account features must be enabled for your account.

- Virtual routing and forwarding (VRF).
- Service endpoints. Enabling service endpoints means that all users in the account can connect to private network endpoints.

To enable VRF, you create a support case. To enable service endpoints, you use the IBM Cloud CLI. For more information about how to enable your account, see [Enabling VRF and service endpoints](#).

Setting a private endpoint

After your account is enabled for VRF and service endpoints, you can add a private network endpoint to a service instance.

A service instance can have a private network endpoint, a public network endpoint, or both.

- Public: A service endpoint on the IBM Cloud public network.
- Private: A service endpoint that is accessible only on the IBM Cloud private network with no access from the public internet.
- Both public and private: Service endpoints that allow access over both networks.

Adding a private network endpoint

You add a private endpoint to a paid service instance from the service details page if you have a Manager or Writer service access role.

1. Go to your [Resource list](#).

2. Click the name of a service instance that is on a paid plan. Lite plans do not support private network endpoints.
3. In the service details page, click the **Manage** tab.
4. Click **Add private network endpoint**.

Viewing your endpoint URL

The service endpoint URLs are different for private and public network endpoints. You can view the URL for an endpoint from the service details page.

1. Go to your Resource list.
2. Click the name of a service instance that has a private network endpoint.
3. In the service details page, click the **Manage** tab, and then click **Private Network Endpoint**.

What to do next

- Configure your account for VRF and Service endpoints.
- Modify your applications to use the new service endpoint URL.
- Read more about service endpoints.

Virtual Private Endpoints

IBM Cloud

IBM Cloud® Virtual Private Endpoints (VPE) for VPC enables you to connect to supported IBM Cloud® services from your VPC network by using the IP addresses of your choosing, allocated from a subnet within your VPC. See more details here.

Note: This document applies to watsonx Assistant, Discovery, Speech to Text, and Text to Speech. Virtual Private Endpoints (VPEs) are available for these services in the Dallas, Washington, Frankfurt, London, Sydney, and Tokyo locations.

Prerequisites

- Have a Virtual Private Cloud (VPC)
- Have private endpoints enabled for your service instance

Instructions

- Create a VPE Gateway (VPEG) through the UI, CLI, or API.
- Creation of the VPEG is confirmed when an IP address is set in the details view of the VPEG page in the UI, CLI output of the `endpoint-gateway` command, or API details call.

You can verify by running `nslookup <endpoint>` on the private service endpoint of the Watson service from your VPC, for example:

```
# nslookup api.private.us-south.assistant.watson.cloud.ibm.com
Server:   127.0.0.53
Address:  127.0.0.53#53

Non-authoritative answer:
Name:   api.private.us-south.assistant.watson.cloud.ibm.com
Address: 10.240.0.9   <---- your VPE IP address
```

To make requests using the assigned IP address instead of just the private service endpoint as suggested here, you must do your own hostname resolution. For example:

```
$ curl -X POST "https://api.private.us-south.assistant.watson.cloud.ibm.com/v2/assistants" --connect ::10.240.0.9
```

```
$ curl -X POST "https://api.private.us-south.assistant.watson.cloud.ibm.com/v2/assistants" --resolve api.private.us-
south.assistant.watson.cloud.ibm.com:443:10.240.0.9
```

Additional links

- IBM Cloud VPC
- VPE FAQ

- Accessing the VPE after setup
- Viewing Details of a VPE Gateway
- VPE Limitations
- Full VPC CLI reference

Service background

The science behind the service

As Pioneering Speech Recognition describes, IBM® has been at the forefront of speech recognition research since the early 1960s. For example, Bahl, Jelinek, and Mercer (1983) describes the basic mathematical approach to speech recognition that is employed in essentially all modern speech recognition systems. And Jelinek (1985) describes the creation of the first real-time large vocabulary speech recognition system for dictation. This paper also describes problems that are still unsolved research topics today.

IBM continues this rich tradition of research and development with the IBM Watson® Speech to Text service. IBM has demonstrated industry-record speech recognition accuracy on the public benchmark data sets for Conversational Telephone Speech (CTS) (Saon and others, 2017) and Broadcast News (BN) transcription (Thomas and others, 2019). IBM leveraged neural networks for language modeling (Kurata and others, 2017a, and Kurata and others, 2017b), in addition to demonstrating the effectiveness of acoustic modeling.

The following announcements summarize IBM's recent speech recognition accomplishments:

- Reaching new records in speech recognition
- IBM Breaks Industry Record for Conversational Speech Recognition by Extending Deep Learning Technologies
- IBM Sets New Transcription Performance Milestone on Automatic Broadcast News Captioning
- IBM Research AI Advances Speaker Diarization in Real Use Cases
- Advancing RNN Transducer Technology for Speech Recognition

These accomplishments contribute to further advance IBM's speech services. Recent ideas that best fit the cloud-based Speech to Text service include

- *For language modeling,* IBM leverages a neural network-based language model to generate training text (Suzuki and others, 2019).
- *For acoustic modeling,* IBM uses a fairly compact model to accommodate the resource limitations of the cloud. To train this compact model, IBM uses "teacher-student training / knowledge distillation." Large and strong neural networks such as Long Short-Term Memory (LSTM), VGG, and the Residual Network (ResNet) are first trained. The output of these networks is then used as teacher signals to train a compact model for actual deployment (Fukuda and others, 2017).

To further push the envelope, IBM also focuses on end-to-end modeling. For example, it has established a strong modeling pipeline for direct acoustic-to-word models (Audhkhasi and others, 2017, and Audhkhasi and others, 2018) that it is now further improving (Saon and others, 2019). It is also making efforts to create compact end-to-end models for future deployment on the cloud (Kurata and Audhkhasi, 2019).

For more information about the scientific research behind the service, see the documents that are listed in Research references.

Research references

For more information about the research behind the IBM Watson® Speech to Text service, see the following documents. IBM® researchers wrote or contributed to all of these papers.

1. Audhkhasi, Kartik, Bhuvana Ramabhadran, George Saon, Michael Picheny, and David Nahamoo. *Direct Acoustics-to-Word Models for English Conversational Speech Recognition.* Proceedings of Interspeech 2017 (August 2017): pp. 959-963.
2. Audhkhasi, Kartik, Brian Kingsbury, Bhuvana Ramabhadran, George Saon, and Michael Picheny. *Building competitive direct acoustics-to-word models for English conversational speech recognition.* Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2018).
3. Bahl, Lalit R., Frederick Jelinek, and Robert L. Mercer. *A Maximum Likelihood Approach to Continuous Speech Recognition.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 5(2) (March 1983): pp. 179-190.
4. Fukuda, Takashi, Masayuki Suzuki, Gakuto Kurata, Samuel Thomas, Jia Cui, and Bhuvana Ramabhadran. *Efficient Knowledge Distillation from an Ensemble of Teachers.* Proceedings of Interspeech 2017 (August 2017): pp. 3697-3701.
5. Graves, Alex. *Sequence Transduction with Recurrent Neural Networks.* International Conference of Machine Learning (ICML), Workshop on Representation Learning (November 2012).
6. Hinton, Geoffrey, Li Deng, Dong Yu, George E. Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N. Sainath, and Brian Kingsbury. *Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups.* Signal Processing Magazine, IEEE, Vol. 29(6) (November 2012): pp. 82-97.
7. Jelinek, Frederick. *The Development of an Experimental Discrete Dictation Recognizer.* Proceedings of the IEEE, Vol. 73(11) (November 1985): pp. 1616-1624.
8. Kurata, Gakuto, Abhinav Sethy, Bhuvana Ramabhadran, and George Saon. *Empirical Exploration of Novel Architectures and Objectives for Language Models.* Proceedings of Interspeech 2017 (August 2017): pp. 279-283.
9. Kurata, Gakuto, Bhuvana Ramabhadran, George Saon, and Abhinav Sethy. *Language Modeling with Highway LSTM.* Proceedings of IEEE Automatic Speech Recognition and Understanding Workshop (ASRU) (2017).

10. Kurata, Gakuto, and Kartik Audhkhasi. *Guiding CTC Posterior Spike Timings for Improved Posterior Fusion and Knowledge Distillation.* Accepted by Interspeech 2019.

11. Padmanabhan, Mukund, and Michael Picheny. *Large-Vocabulary Speech Recognition Algorithms.* Computer, Vol. 35(4) (2002): pp. 42-50.

12. Picheny, Michael, David Nahamoo, Vaibhava Goel, Brian Kingsbury, Bhuvana Ramabhadran, Steven J. Rennie, and George Saon. *Trends and Advances in Speech Recognition.* IBM Journal of Research and Development, Vol. 55(5) (October 2011): pp. 2:1-2:18.

13. Saon, George, Zoltan Tueske, Daniel Bolanos, and Brian Kingsbury. *Advancing RNN Transducer Technology for Speech Recognition* Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (March 2021).

14. Saon, George, Zoltán Tüsky, and Kartik Audhkhasi. *Alignment-Length Synchronous Decoding for RNN Transducer.* Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (May 2020).

15. Saon, George, Zoltan Tueske, Kartik Audhkhasi, and Brian Kingsbury. *Sequence Noise Injected Training for End-to-end Speech Recognition.* Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (May 2019).

16. Saon, George, Gakuto Kurata, Tom Sercu, Kartik Audhkhasi, Samuel Thomas, Dimitrios Dimitriadis, Xiaodong Cui, Bhuvana Ramabhadran, Michael Picheny, Lynn-Li Lim, Bergul Roomi, and Phil Hall. *English Conversational Telephone Speech Recognition by Humans and Machines.* Proceedings of Interspeech 2017 (August 2017): pp. 132-136.

17. Saon, George, Tom Sercu, Steven Rennie, and Hong-Kwang J. Kuo. *The IBM 2016 English Conversational Telephone Speech Recognition System.* Submitted to Interspeech 2016 (2016).

18. Saon, George, Hong-Kwang J. Kuo, Steven Rennie, and Michael Picheny. *The IBM 2015 English Conversational Telephone Speech Recognition System.* Submitted to Interspeech 2015 (2015).

19. Soltau, Hagen, George Saon, and Tara N. Sainath. *Joint Training of Convolutional and Non-Convolutional Neural Networks.* Proceedings of the IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP), Florence, Italy (May 2014): pp. 5572-5576.

20. Suzuki, Masayuki, Nobuyasu Itoh, Tohru Nagano, Gakuto Kurata, and Samuel Thomas. *Improvements to N-gram Language Model Using Text Generated from Neural Language Model.* Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (May 2019).

21. Thomas, Samuel, Masayuki Suzuki, Yinghui Huang, Gakuto Kurata, Zoltan Tuske, George Saon, Brian Kingsbury, and Michael Picheny. *English Broadcast News Speech Recognition by Humans and Machines.* Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (2019).

22. Kartik Audhkhasi, George Saon, Zoltán Tüsky, Brian Kingsbury and Michael Picheny. *Forget a Bit to Learn Better: Soft Forgetting for CTC-Based Automatic Speech Recognition.* Proceedings of Interspeech 2019 (September 2019): pp. 2618-2622.

High availability and disaster recovery

IBM Cloud

The IBM Watson® Speech to Text service is highly available within any IBM Cloud location (for example, Dallas or Washington, DC). However, recovering from potential disasters that affect an entire location requires planning and preparation.

You are responsible for understanding your configuration, customization, and usage of the service. You are also responsible for being ready to re-create an instance of the service in a new location and to restore your data in any location.

High availability

The Speech to Text service supports high availability with no single point of failure. The service achieves high availability automatically and transparently by means of the multi-zone region (MZR) feature provided by IBM Cloud.

IBM Cloud enables multiple zones that do not share a single point of failure within a single location. It also provides automatic load balancing across the zones within a region.

Disaster recovery

Disaster recovery can become an issue if an IBM Cloud location experiences a significant failure that includes the potential loss of data. Because MZR is not available across locations, you must wait for IBM to bring a location back online if it becomes unavailable. If underlying data services are compromised by the failure, you must also wait for IBM to restore those data services.

In the event of a catastrophic failure, IBM might not be able to recover data from database backups. In this case, you need to restore your data to return your service instance to its most recent state. You can restore the data to the same or to a different location.

Your disaster recovery plan includes knowing, preserving, and being prepared to restore all data that is maintained on IBM Cloud. This includes data from custom language models, custom acoustic models, and asynchronous speech recognition requests.

Note: Re-creating custom models, especially custom acoustic models, from saved data can take a significant amount of time. Maintaining parallel service configurations in multiple locations can eliminate the turnaround time associated with disaster recovery.

Disaster recovery for custom language models

For custom language models, you need to maintain and be prepared to re-create and restore your custom language models and their corpora, grammars, and custom words. You also need to be prepared to retrain your custom language models on their data and words resources.

Backing up custom language models

Preserve the following information about your custom language models:

- A list of all of your custom language models and their definitions. To list information about your custom models:
  - Use the `GET /v1/customizations` method to list information about all custom models.
  - Use the `GET /v1/customizations/{customization_id}` method to list information about a specified custom model.

  For more information, see Listing custom language models.

- Copies of all corpus text files that you add to your custom language models. To list information about the corpora for your custom models:
  - Use the `GET /v1/customizations/{customization_id}/corpora` method to list all corpora for a custom model.
  - Use the `GET /v1/customizations/{customization_id}/corpora/{corpus_name}` method to list information about a specified corpus for a custom model.

  For more information, see Listing corpora for a custom language model.

- Copies of all grammar files that you add to your custom language models. To list information about the grammars for your custom models:
  - Use the `GET /v1/customizations/{customization_id}/grammars` method to list information about all grammars for a custom model.
  - Use the `GET /v1/customizations/{customization_id}/grammars/{grammar_name}` method to list information about a specified grammar for a custom model.

  For more information, see Listing grammars for a custom language model.

- Information about all custom words, including their sounds-like and display-as definitions, that you add directly to your custom language models. To list information about the out-of-vocabulary (OOV) words for your custom models:

  - Use the `GET /v1/customizations/{customization_id}/words` method to list information about the words from a custom model. You can use the `word_type` parameter to list `all` words from a model, words added directly by the `user`, words extracted from `corpora`, or words recognized by `grammars`.
  - Use the `GET /v1/customizations/{customization_id}/words/{word_name}` method to list information about a specified word from a custom model.

  For more information, see Listing custom words from a custom language model.

It is a best practice to preserve this information in a format that you can use to re-create your custom language models in the event of a failure. Actively maintaining the information about your custom models and their data, and preparing the calls listed in the following section ahead of time, can enable you to recover as quickly as possible.

Restoring custom language models

If you need to recover from a disaster, you can use the backup information to re-create your custom language models and their data:

1. To re-create your custom language models, use the `POST /v1/customizations` method. For more information, see Create a custom language model.
2. To add the corpus text files to your custom models, use the `POST /v1/customizations/{customization_id}/corpora/{corpus_name}` method. For more information, see Add a corpus to the custom language model.
3. To add the grammar files to your custom models, use the `POST /v1/customizations/{customization_id}/grammars/{grammar_name}` method. For more information, see Add a grammar to the custom language model.
4. To add multiple words to your custom models, use the `POST /v1/customizations/{customization_id}/words` method. To add single words to your custom models, use the `PUT /v1/customizations/{customization_id}/words/{word_name}` method. For more information, see Add words to the custom language model.
5. To train your custom models once you restore your corpora, grammars, and custom words, use the `POST /v1/customizations/{customization_id}/train` method. For more information, see Train the custom language model.

The methods that you use to add corpora, grammars, and words and to train a custom language model are asynchronous. You need to monitor the requests until they complete.

You can incrementally add data and train your custom language models rather than adding all data before training the models. For example, you can add your corpora and then train your models before you add grammars and individual words. You can also incrementally add and train your models on individual corpus text files, grammar files, and groups of custom words.

Disaster recovery for custom acoustic models

For custom acoustic models, you need to maintain and be prepared to re-create and restore your custom acoustic models and their audio resources. You also need to be prepared to retrain your custom acoustic models on their audio resources.

Backing up custom acoustic models

Preserve the following information about your custom acoustic models:

- A list of all of your custom acoustic models and their definitions. To list information about your custom models:

  - Use the `GET /v1/acoustic_customizations` method to list information about all custom models.
  - Use the `GET /v1/acoustic_customizations/{customization_id}` method to list information about a specified custom model.

  For more information, see Listing custom acoustic models.

- Copies of all audio resources, both individual audio files and archive files, that you add to your custom acoustic models. To list information about the audio resources for your custom models:

  - Use the `GET /v1/acoustic_customizations/{customization_id}/audio` method to list information about all audio resources for a custom model.
  - Use the `GET /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method to list information about a specified audio resource for a custom model.

  For more information, see Listing audio resources for a custom acoustic model.

It is a best practice to preserve this information in a format that you can use to re-create your custom acoustic models in the event of a failure. Actively maintaining the information about your custom models and their audio resources, and preparing the calls listed in the following section ahead of time, can enable you to recover as quickly as possible.

Restoring custom acoustic models

If you need to recover from a disaster, you can use the backup information to re-create your custom acoustic models and their data:

1. To re-create your custom acoustic models, use the `POST /v1/acoustic_customizations` method. For more information, see Create a custom acoustic model.
2. To add the audio resources to your custom models, use the `POST /v1/acoustic_customizations/{customization_id}/audio/{audio_name}` method. For more information, see Add audio to the custom acoustic model.
3. To train your custom models once you restore your audio resources, use the `POST /v1/acoustic_customizations/{customization_id}/train` method. For more information, see Train the custom acoustic model.

The methods that you use to add audio resources and to train a custom acoustic model are asynchronous. You need to monitor the requests until they complete.

You can incrementally add audio resources and train your custom acoustic models rather than adding all data before training the models. For example, you can add your audio resources one at a time or in groups, training your models on subsets of the audio resources rather than on all of the audio at one time.

Disaster recovery for asynchronous speech recognition jobs

For speech recognition with the asynchronous HTTP interface, you need to maintain the following information:

- All callback URLs that you allowlist for use with the asynchronous interface. If a failure occurs, you might need to use the `POST /v1/register_callback` method to re-register the URLs. The method returns an appropriate response if a URL is already allowlisted.
- Copies of the audio files that you submit to the asynchronous interface for speech recognition. If a failure occurs before you receive or retrieve the results of a completed asynchronous job, you need to use the `POST /v1/recognitions` method to resubmit the audio files when the service is restored. Once you have the results of a completed asynchronous job, you no longer need to maintain the audio files.

For more information, see The asynchronous HTTP interface. As with the backup data for custom models, you can actively preserve this information and be prepared to reissue the necessary requests ahead of time.

Activity Tracker events

IBM Cloud

⚠ **Important:** As of 28 March 2024, the IBM Cloud Activity Tracker service is deprecated and will no longer be supported as of 30 March 2025. Customers will need to migrate to IBM Cloud Logs before 30 March 2025. During the migration period, customers can use IBM Cloud Activity Tracker along with IBM Cloud Logs. Activity tracking events are the same for both services. For information about migrating from IBM Cloud Activity Tracker to IBM Cloud Logs and running the services in parallel, see migration planning.

As a security officer, auditor, or manager, you can use the Activity Tracker service to track how users and applications interact with IBM Watson® Speech to Text in IBM Cloud®.

IBM Cloud Activity Tracker records user-initiated activities that change the state of a service in IBM Cloud. You can use this service to investigate abnormal activity and critical actions and to comply with regulatory audit requirements. In addition, you can be alerted about actions as they happen. The events that are collected comply with the Cloud Auditing Data Federation (CADF) standard. For more information, see the tutorial Getting started with IBM Cloud Activity Tracker.

Locations where activity tracking events are generated

Locations where activity tracking events are sent to IBM Cloud Activity Tracker hosted event search

IBM Watson® Speech to Text in IBM Cloud® sends activity tracking events to IBM Cloud Activity Tracker hosted event search in the regions that are indicated in the following table.

Dallas (us-south)	Washington (us-east)	Toronto (ca-tor)	Sao Paulo (br-sao)
No	No	No	No

Regions where activity tracking events are sent in Americas locations

Tokyo (jp-tok)	Sydney (au-syd)	Osaka (jp-osa)	Chennai (in-che)
No	No	No	No

Regions where activity tracking events are sent in Asia Pacific locations

Frankfurt (eu-de)	London (eu-gb)	Madrid (eu-es)
Yes	No	No

Regions where activity tracking events are sent in Europe locations

Language model customization events

The following tables list the Speech to Text actions for language model customization that generate events.

Create events

Action	Description
`speech-to-text.custom-language-model.create`	Create a custom language model (`POST /v1/customizations`).
`speech-to-text.custom-language-model-word-list.create`	Create a word list for a custom language model (`POST /v1/customizations/{customization_id}/words`).
`speech-to-text.custom-language-model-word.create`	Create a word for a custom language model (`PUT /v1/customizations/{customization_id}/words/{word_name}`).
`speech-to-text.custom-language-model-corpus.create`	Create a corpus for a custom language model (`POST /v1/customizations/{customization_id}/corpora/{corpus_name}`).

`speech-to-text.custom-language-model-grammar.create`	Create a grammar for a custom language model (`POST /v1/customizations/{customization_id}/grammars/{grammar_name}`).

Table 1. Language model customization .create actions that generate events

Read events

Action	Description
`speech-to-text.custom-language-model-list.read`	Read a list of custom language models (`GET /v1/customizations`).
`speech-to-text.custom-language-model.read`	Read a custom language model (`GET /v1/customizations/{customization_id}`).
`speech-to-text.custom-language-model-word-list.read`	Read a word list of a custom language model (`GET /v1/customizations/{customization_id}/words`).
`speech-to-text.custom-language-model-word.read`	Read a word for a custom language model (`GET /v1/customizations/{customization_id}/words/{word_name}`).
`speech-to-text.custom-language-model-corpus-list.read`	Read a corpus list of a custom language model (`GET /v1/customizations/{customization_id}/corpora`).
`speech-to-text.custom-language-model-corpus.read`	Read a corpus of a custom language model (`GET /v1/customizations/{customization_id}/corpora/{corpus_name}`).
`speech-to-text.custom-language-model-grammar-list.read`	Read a grammar list of a custom language model (`GET /v1/customizations/{customization_id}/grammars`).
`speech-to-text.custom-language-model-grammar.read`	Read a grammar of a custom language model (`GET /v1/customizations/{customization_id}/grammars/{grammar_name}`).

Table 2. Language model customization .read actions that generate events

Delete events

Action	Description
`speech-to-text.custom-language-model.delete`	Delete a custom language model (`DELETE /v1/customizations/{customization_id}`).
`speech-to-text.custom-language-model-word.delete`	Delete a word from a custom language model (`DELETE /v1/customizations/{customization_id}/words/{word_name}`).
`speech-to-text.custom-language-model-corpus.delete`	Delete a corpus from a custom language model (`DELETE /v1/customizations/{customization_id}/corpora/{corpus_name}`).
`speech-to-text.custom-language-model-grammar.delete`	Delete a grammar from a custom language model (`DELETE /v1/customizations/{customization_id}/grammars/{grammar_name}`).

Table 3. Language model customization .delete actions that generate events

Train event

Action	Description
`speech-to-text.custom-language-model.train`	Train a custom language model (`POST /v1/customizations/{customization_id}/train`).

Table 4. Language model customization .train action that generates an event

Reset event

Action	Description
`speech-to-text.custom-language-model.reset`	Reset a custom language model (`POST /v1/customizations/{customization_id}/reset`).

Table 5. Language model customization .reset action that generates an event

Upgrade event

Action	Description
`speech-to-text.custom-language-model.upgrade`	Upgrade a custom language model (`POST /v1/customizations/{customization_id}/upgrade_model`).

Table 6. Language model customization .upgrade action that generates an event

Acoustic model customization events

The following tables list the Speech to Text actions for acoustic model customization that generate events.

Create events

Action	Description
`speech-to-text.custom-acoustic-model.create`	Create a custom acoustic model (`POST /v1/acoustic_customizations`).
`speech-to-text.custom-acoustic-model-audio.create`	Create an audio for a custom acoustic model (`POST /v1/acoustic_customizations/{customization_id}/audio/{audio_name}`).

Table 7. Acoustic model customization .create actions that generate events

Read events

Action	Description
`speech-to-text.custom-acoustic-model-list.read`	Read a list of custom acoustic models (`GET /v1/acoustic_customizations`).
`speech-to-text.custom-acoustic-model.read`	Read a custom acoustic model (`GET /v1/acoustic_customizations/{customization_id}`).
`speech-to-text.custom-acoustic-model-audio-list.read`	Read an audio list of a custom acoustic model (`GET /v1/acoustic_customizations/{customization_id}/audio`).
`speech-to-text.custom-acoustic-model-audio.read`	Read an audio of a custom acoustic model (`GET /v1/acoustic_customizations/{customization_id}/audio/{audio_name}`).

Table 8. Acoustic model customization .read actions that generate events

Delete events

Action	Description
`speech-to-text.custom-acoustic-model.delete`	Delete a custom acoustic model (`DELETE /v1/acoustic_customizations/{customization_id}`).
`speech-to-text.custom-acoustic-model-audio.delete`	Delete an audio from a custom acoustic model (`DELETE /v1/acoustic_customizations/{customization_id}/audio/{audio_name}`).

Table 9. Acoustic model customization .delete actions that generate events

Train event

Action	Description
`speech-to-text.custom-acoustic-model.train`	Train a custom acoustic model (`POST /v1/acoustic_customizations/{customization_id}/train`).

Table 10. Acoustic model customization .train action that generates an event

Reset event

Action	Description
`speech-to-text.custom-acoustic-model.reset`	Reset a custom acoustic model (`POST /v1/acoustic_customizations/{customization_id}/reset`).

Table 11. Acoustic model customization .reset action that generates an event

Upgrade event

Action	Description
`speech-to-text.custom-acoustic-model.upgrade`	Upgrade a custom acoustic model (`POST /v1/acoustic_customizations/{customization_id}/upgrade_model`).

Table 12. Acoustic model customization .upgrade action that generates an event

Asynchronous HTTP interface events

The following tables list the Speech to Text actions for the asynchronous HTTP interface that generate events.

Create events

Action	Description
`speech-to-text.async-recognition-job.create`	Create an asynchronous recognition job (`POST /v1/recognitions`).
`speech-to-text.async-recognition-notification-url.create`	Create an asynchronous recognition notification URL (`POST /v1/register_callback`).

Table 13. Asynchronous HTTP recognition .create actions that generate events

Read events

Action	Description
`speech-to-text.async-recognition-job-list.read`	Read an asynchronous recognition job list (`GET /v1/recognitions`).
`speech-to-text.async-recognition-job.read`	Read an asynchronous recognition job (`GET /v1/recognitions/{id}`).

Table 14. Asynchronous HTTP recognition .read actions that generate events

Delete events

Action	Description
`speech-to-text.async-recognition-job.delete`	Delete an asynchronous recognition job (`DELETE /v1/recognitions/{id}`).
`speech-to-text.async-recognition-notification-url.delete`	Delete an asynchronous recognition notification URL (`POST /v1/unregister_callback`).

Table 15. Asynchronous HTTP recognition .delete actions that generate events

GDPR event

The following table lists the Speech to Text action for General Data Protection Regulation (GDPR) that generates an event.

Action	Description
`speech-to-text.gdpr-user-data.delete`	Delete information for a user (`DELETE /v1/user_data`).

Table 16. GDPR .delete action that generates an event

Where to view events

Events that are generated by an instance of the Speech to Text service are automatically forwarded to the IBM Cloud Activity Tracker service instance that is available in the same location.

IBM Cloud Activity Tracker can have only one instance per location. To view events, you must access the web user interface (UI) of the IBM Cloud Activity Tracker service in the same location where your service instance is available. For more information, see Navigating to the UI.

Watson SDKs

SDKs abstract much of the complexity associated with application development. By providing programming interfaces in languages that you already know, they can help you get up and running quickly with IBM Watson services.

Supported SDKs

The following Watson SDKs are supported by IBM:

- Java SDK
- Node.js SDK
- Python SDK
- .NET SDK

Tip: The API reference for each service includes information and examples for these SDKs.

Community SDKs

The following SDKs are available from the Watson community of developers:

- ABAP SDK for IBM Watson, using SAP NetWeaver
- Android SDK
- Go SDK
- Ruby SDK
- Salesforce SDK
- Swift SDK
- Unity SDK

SDK updates and deprecation

The supported Watson SDKs are updated according to the following guidelines.

Semantic versioning

Supported Watson SDKs adhere to semantic versioning with releases labeled as `{major}.{minor}.{patch}`.

Release frequency

SDKs are released independently and might not update on the same schedule.

- The current releases of the Watson SDKs are updated on a 2- to 6-week schedule. These releases are either minor updates or patches that do not include breaking changes. You can update to any version of the SDK with the same major version number.
- Major updates that might include breaking changes are released approximately every 6 months.

Deprecated release

When a major version is released, support continues on the previous major release for 12 months in a deprecation period. The deprecated release might be updated with bug fixes, but no new features will be added and documentation might not be available.

Obsolete release

After the 12-month deprecation period, a release is obsolete. The release might be functional but is unsupported and not updated. Update to the current release.

Known limitations

The Speech to Text service has the following known limitations. These issues apply to service functionality that spans releases for all platforms. For information about known limitations specific to Speech to Text for IBM Cloud Pak for Data version 4.6.x, see Limitations and known issues in Watson Speech services.

Issue: Interim results for previous-generation models

27 April 2021: When you use previous-generation models with the WebSocket interface, if you request both speaker labels and interim results, the speaker labels response includes an extra object. The final results duplicate the object for the last speaker label. Instead of appearing once with `"final": true`, the object appears twice: once with `"final": false` and once with `"final": true`.

For example, a WebSocket request that includes both speaker labels and interim results sends a `start` message like the following:

```
var message = {
  action: 'start',
  speaker_labels: true,
  interim_results: true
};
websocket.send(JSON.stringify(message));
```

To indicate final results for speaker labels, the service is supposed to return a response of the following form:

```
{
  "speaker_labels": [
    . . .
    {
      "from": 1.01,
      "to": 1.75,
      "speaker": 0,
      "confidence": 0.75,
      "final": false
    },
    {
      "from": 1.76,
      "to": 2.50,
      "speaker": 1,
      "confidence": 0.80,
      "final": true
    }
  ]
}
```

Instead, the service returns final results like the following. Note that the object for the last speaker label, which begins at `1.76` and ends at `2.50`, is returned twice. The `"final": true` indication is associated with the second instance of the label.

```
{
  "speaker_labels": [
    . . .
    {
      "from": 1.01,
      "to": 1.75,
      "speaker": 0,
      "confidence": 0.75,
      "final": false
    },
    {
      "from": 1.76,
      "to": 2.50,
      "speaker": 1,
      "confidence": 0.80,
      "final": false
    }
  ]
}{
  "from": 1.76,
  "to": 2.50,
  "speaker": 1,
  "confidence": 0.80,
  "final": true
}
```

For more information about using speaker labels with interim results, see [Requesting interim results for speaker labels]().

Issue: Supported features for speaker labels is always `true`

6 August 2020: The `GET /v1/models` and `GET /v1/models/{model_id}` methods list information about language models. Under `supported_features`, the `speaker_labels` field indicates whether you can use the `speaker_labels` parameter with a model. At this time, the field returns `true` for all models.

However, speaker labels are supported as beta functionality only for US English, Australian English, German, Japanese, Korean, and Spanish (both broadband and narrowband models) and UK English (narrowband model only). Speaker labels are not supported for any other models. Do not rely on the field to identify which models support speaker labels.

For more information about speaker labels and supported models, see [Speaker labels]().

Issue: The `progress` field for custom models is not continuous

The following methods include a `progress` field in the information that they return for a custom model:

- `GET /v1/customizations`
- `GET /v1/customizations/{customization_id}`
- `GET /v1/acoustic_customizations`
- `GET /v1/acoustic_customizations/{customization_id}`

The field returns the progress for a training or upgrade operation. Currently, the value of the field is `100` if the status is `available`; otherwise, it is `0`.

When you monitor the training or upgrading of a custom model, poll the value of the `status` field, not the value of the `progress` field. If the operation fails for any reason, the value of the `status` field changes to reflect the failure; the value of the `progress` field remains `0`.

Usage FAQs

FAQs for IBM Watson® Speech to Text include questions about speech recognition, audio transmission, customization, and other topics. To find all FAQs for IBM Cloud®, see the FAQ library.

How do I access my service credentials?

How you access your service credentials depends on whether you are using Speech to Text with IBM Cloud® or IBM Cloud Pak® for Data. For more information about obtaining your credentials for both versions, see Before you begin in the getting started tutorial.

Once you have your service credentials, see the following topics for information about authenticating to the service:

- Authenticating to IBM Cloud
- Authenticating to IBM Cloud Pak for Data

What languages does the service support?

The Speech to Text service supports large speech models, previous-generation and next-generation languages and models. Most languages support both broadband/multimedia and narrowband/telephony models, which have minimum sampling rates of 16 kHz and 8 kHz, respectively. For more information about the available models and the features they support for all languages, see the following topics:

- Large speech languages and models
- Previous-generation languages and models
- Next-generation languages and models

What are the input audio formats?

The service supports many audio formats (MIME types). Different formats support different sampling rates and other characteristics. By using a format that supports compression, you can maximize the amount of audio data that you can send with a request. For more information about the supported audio formats, see the following topics:

- Audio terminology and characteristics
- Supported audio formats

How much audio data can I submit to the service?

The amount of audio that you can submit with a single speech recognition request depends on the interface that you are using:

- The WebSocket and synchronous HTTP interfaces accept a maximum of 100 MB of audio data.
- The asynchronous HTTP interface accepts a maximum of 1 GB of audio data.

For more information, see Recognizing speech with the service.

Can I transcribe speech from video files?

You cannot transcribe speech from a multimedia file that contains both audio and video. To transcribe speech from a video file, you must separate the audio data from the video data. For more information, see Transcribing speech from video files.

How can I improve transcription accuracy?

The Speech to Text service offers a customization interface that provides many features and options to improve the speech recognition capabilities of the supported base language models:

- If you are transcribing audio for a specific domain, you can create custom language models to expand and tailor a base model's vocabulary to include domain-specific terminology. When you use custom language models, you can also create and incorporate custom grammars to restrict the words that the service can recognize from your model's vocabulary. Language model customization is supported for large speech models, previous- and next-generation models. For more information, see Creating a custom language model and Adding a grammar to a custom language model.
- If you are transcribing audio with unique characteristics (such as speaker accents, telephone conversations, or background noise), you can create a custom acoustic model to adapt a base model for your environment and speakers. Acoustic model customization is supported only for previous-generation models. For more information, see Creating a custom acoustic model.
- You can also use custom acoustic and custom language models together. If transcriptions or related corpora are available for your audio, you can use that data to create a complementary custom language model to further improve the quality of speech recognition based on your custom acoustic

model. Grammars are supported only for previous-generation models. For more information, see Using custom acoustic and custom language models together.

How many words can I add to a custom language model?

You can add a maximum of 90 thousand out-of-vocabulary (OOV) words to a custom language model from all sources. You can add a maximum of 10 million total words to a custom language model from all sources. But many factors contribute to the amount of data that you need for an effective custom language model. Although it is not possible to provide the exact number of words that you need to add for any custom model or application, even adding a few words to a custom model can improve speech recognition. For more information about limits on the number of words that you can add and for other factors that affect the amount of data that you need, see How much data do I need?.

How does custom model upgrading work?

When a new version of a previous-generation base model is released to improve the quality of speech recognition, you must upgrade any custom language and custom acoustic models that are based on that model to take advantage of the updates. When you upgrade a custom model, you do not need to upgrade its resources individually. The service upgrades the resources automatically. Custom model upgrading applies only to previous-generation models.

- For more information about upgrading a custom model, see Upgrading custom models.
- For more information about using an upgraded custom model for a speech recognition request, see Using upgraded custom models for speech recognition.

Can the Speech to Text service transcribe numbers as digits instead of strings?

For US English, Brazilian Portuguese, French, German and the US English Medical model, you can use the new version of smart formatting feature available. For more information, see New Smart formatting.

For Japanese, and Spanish audio, you can use smart formatting to convert certain strings, such as digits and numbers, to more conventional representations. Smart formatting is beta functionality. For more information, see Smart formatting.

Pricing FAQs

IBM Cloud

The IBM Watson® Speech to Text service is available at three pricing plans: Lite, Plus, and Premium. The following FAQs provide an overview of the pricing plans. For more information, see the Speech to Text service in the IBM Cloud® Catalog.

Note: The Standard plan is no longer available for purchase by users. The Standard plan continues to be available to its existing users indefinitely. For more information, see Can I continue to use the Speech to Text Standard plan? For new users, read about the new Plus and Premium plans below.

What is the price for using the Speech to Text Lite plan?

The Lite plan lets you get started with 500 minutes per month of speech recognition at no cost. You can use any available model for speech recognition. The Lite plan does not provide access to customization. You must use a paid plan to use customization.

The Lite plan is intended for any user who wants to try out the service before committing to a purchase. For more information, see the Speech to Text service in the IBM Cloud® Catalog. Services that are created with the Lite plan are deleted after 30 days of inactivity.

What is the price for using the Speech to Text Plus plan?

The Plus plan provides access to all of the service's features:

- Use of all available models (same as the Lite plan).
- Unlimited creation and use of custom language and custom acoustic models at no extra charge.
- A maximum of one hundred concurrent transcription requests from all interfaces, WebSocket and HTTP, combined.

The plan uses a simple tiered pricing model to give high-volume users further discounts as they use the service more heavily. Pricing is based on the aggregate number of minutes of audio that you recognize per month:

- Users who recognize 1 to 999,999 minutes of audio in a given month pay $0.02 (USD) per minute of audio for that month.
- Users who recognize at least 1,000,000 minutes of audio in a given month pay $0.01 (USD) per minute of audio for that month.

The Plus plan is intended for small businesses. It is also a good choice for large enterprises that want to develop and test larger applications before considering moving to a Premium plan. For more information, see the Speech to Text service in the IBM Cloud® Catalog.

Can I continue to use the Speech to Text Standard plan?

The Standard plan is no longer available for purchase by new users. But existing users of the Standard plan can continue to use the plan indefinitely with no change in their pricing. Their API settings and custom models remain unaffected.

Existing users can also choose to upgrade to the new Plus plan by visiting the IBM Cloud® Catalog. They will continue to have access to all of their settings and custom models after upgrading. And, if they find that the Plus plan does not meet their needs for any reason, they can always downgrade back to their Standard plan.

What pricing plan do I need to use the service's customization interface?

You must have a paid plan (Plus, Standard, or Premium) to use language model or acoustic model customization. Users of the Lite plan cannot use customization. To use customization, users of the Lite plan must upgrade to a paid plan such as the Plus plan.

How do I upgrade from the Lite plan to the Plus plan?

You can upgrade from the Lite plan to the Plus plan, for example, to gain access to customization. To upgrade from the Lite plan to the Plus plan, use the **Upgrade** button in the resource catalog page for your service instance:

- From the resource list, click on your Speech to Text instance to go to the Speech to Text dashboard page.
- From the **Manage** page, click **Upgrade** to move to the Plus plan.
- Follow the steps on the **Plan** page to complete your upgrade.

What does "pricing per minute" mean?

For the Plus plan, pricing is based on the cumulative amount (number of minutes) of audio that you send to the service in any one month. The per-minute price of all audio that you recognize in a month is reduced once you reach the threshold of one million minutes of audio for that month. The price does not depend on how long the service takes to process the audio. (Per-minute pricing is different for the Standard plan.)

For information about pricing for the Plus and Standard plans, see the Speech to Text service in the IBM Cloud® Catalog.

Do you round up to the nearest minute for every call to the API?

IBM does not round up the length of the audio for every API call that the service receives. Instead, IBM aggregates all usage for the month and rounds to the nearest minute at the end of the month. For example, if you send two audio files that are each 30 seconds long, IBM sums the duration of the total audio for that month to one minute.

Am I charged for all audio that I send, even audio that does not include speech?

Yes, all audio that you send to the service contributes to your cumulative minutes of audio. This includes silence and noisy audio that does not contain or otherwise contribute to speech recognition. Because the service must process all audio that it receives, it does not distinguish between the type or quality of audio that you send. For pricing purposes, three seconds of silence is equivalent to three seconds of actual speech.

What advantages do I get by using a Speech to Text Premium plan?

The Premium plan offers developers and organizations all of the capabilities and features of the Plus plan. The plan also offers these additional features:

- The ability to use IBM Watson® services in the IBM Cloud® with data isolation and added security features such as IBM® Key Protect for IBM Cloud® (also referred to as *Bring your own keys* (BYOK)), Service Endpoints, Mutual Authentication, and US Health Insurance Portability and Accountability Act (HIPAA) readiness.
- Your first 150,000 minutes of speech recognition each month at no charge.

To learn more or to make a purchase, contact an IBM representative.

Can I provision a premium instance right now?

No, you must first contact your IBM sales representative to purchase an entitlement and share your requirements. To make a purchase, contact an IBM representative.

Accessibility features for IBM Cloud

Accessibility features assist users who have a disability, such as restricted mobility or limited vision, to use information technology content successfully.

Accessibility features

IBM Cloud® includes the following major accessibility features:

- Keyboard-only operation
- Operations that use a screen reader

IBM Cloud uses the W3C Standard, WAI-ARIA 1.1, to ensure compliance to ensure compliance to US Section 508, Web Content Accessibility Guidelines (WCAG) 2.1, and EN 301 549. To take advantage of accessibility features, use the latest release of your screen reader in combination with the latest Firefox web browser that is supported by this product.

The IBM Cloud online product documentation and the IBM Cloud user interface framework is enabled for accessibility.

Interface information

Review the following information about the IBM Cloud user interface:

- If you are using a screen reader with the IBM Cloud web user interface or product documentation, use the latest version of Firefox with the latest release of the screen reader.
- If you are using keyboard operation only, ensure that your MacOS setting is enabled for `Keyboard navigation: Use keyboard navigation to move focus between controls`.
- The IBM Cloud user interfaces do not have content that flashes 2 - 55 times per second.
- The IBM Cloud web user interfaces rely on cascading style sheets to render content properly and to provide a usable experience. The application provides an equivalent way for low-vision users to use a user's system display settings, including high-contrast mode. You can control font size by using the device or web browser settings.
- The IBM Cloud web user interface includes WAI-ARIA navigational landmarks that you can use to quickly navigate to functional areas in the application.

Related accessibility information

The IBM Cloud web user interface accessibility compliance status is specifically for the IBM Cloud product platform. There are subsections of the user interface that are owned by third-party products or services that host content within the platform, for which the IBM Cloud compliance record does not maintain or own the accessibility compliance status. If you are accessing any user interface or documentation for a service, you must request the compliance statements for that service. For example, if you are using an interface for IBM Cloud Kubernetes Service, the administration console for a local or dedicated environment, or an IoT service, you must request product accessibility information for that interface or documentation.

The IBM Cloud documentation accessibility compliance is specifically for the IBM Cloud core platform information, and it does not extend to any services. The available documentation for IBM Cloud is managed and reported in the IBM Cloud product accessibility information that is available upon request. For compliance status for any service, you must request product accessibility information.

In addition to standard IBM help desk and support websites, IBM has established a TTY telephone service for use by deaf or hard of hearing customers to access sales and support services:

TTY service 800-IBM-3383 (800-426-3383) (within North America)

IBM and accessibility

For more information about the commitment that IBM has to accessibility, see IBM Accessibility.