SPEECHMATICS

Real-time Container 1.0.0

Table of Contents

Real-time Container

Summary

These are the General Availability (GA) release notes for the Real-Time ASR container images. Following languages are supported:

- English (en)
- German (de)
- Spanish (es)
- French (fr)
- Portuguese (pt)
- Japanese (ja)
- Korean (ko)
- Dutch (nl)
- Italian (it)
- Swedish (sv)
- Danish (da)
- Polish (pl)
- Catalan (ca)
- Hindi (hi)
- Russian (ru)
- Mandarin (cmn)
- Norwegian (no)
- Arabic (ar)
- Bulgarian (bg)
- Czech (cs)
- Greek (el)
- Finnish (fi)
- Hungarian (hu)
- Croatian (hr)
- Lithuanian (lt)
- Latvian (lv)
- Romanian (ro)
- Slovak (sk)
- Slovenian (sl)
- Turkish (tr)
- Malay (ms)

Container images are labelled using the following scheme, where language codes adhere the ISO-639 standard:

```
rt-transcriber-<language>:<version>
```

For example,

```
rt-transcriber-en:1.1.0
```

Known Issues

The following are known issues in this release:

Reference	Summary	Workarounds
REQ-	Chinese (cmn) container crashes occasionally	Do not use whitespace characters in

13240	when using certain additional vocabulary	additional vocabulary `sounds_like`

Supported Platforms

Docker 17.06.0+

Installation

Pull the container image from the Speechmatics Docker registry.

Prerequisites

- Docker (17.06.0 or above).
- Login credentials (URL, username and password) for the Speechmatics Docker registry.

Real-time Container Quick Start Guide

This guide will walk you through the steps needed to deploy the Speechmatics Real-time Container ready for transcription.

- Check system requirements
- Pull the Docker Image
- Run the Container

After these steps, the Docker Image can be used to create containers that will transcribe audio files. More information about using the API for real-time transcription is detailed in the Speech API guide.

System Requirements

Speechmatics containerized deployments are built on the Docker platform. At present a separate Docker image is required for each language to be transcribed. Each docker image takes about {{ book.requirements.image_size }} of storage.

A single image can be used to create and run multiple containers concurrently, each running container will require the following resources:

- {{ book.requirements.cpus }}
- {{ book.requirements.memory }} RAM
- {{ book.requirements.storage }} hard disk space

The host machine requires a processor with following minimum specification: Intel® Xeon® CPU E5-2630 v4 (Sandy Bridge) 2.20GHz (or equivalent). This is important because these chipsets (and later ones) support Advanced Vector Extensions (AVX). The machine learning algorithms used by Speechmatics ASR require the performance optimizations that AVX provides. You should also ensure that your hypervisor has AVX enabled.

Architecture

Each container:

- Provides the ability to transcribe speech data in a predefined language from a live stream or a recorded audio file. The container will receive audio input using a WebSocket interface, and will provide the following output:
  - Words in the transcript
  - Word confidence
  - Timing information

- Multiple instances of the container can be run on the same Docker host. This enables scaling of a single language or multiple-languages as required
- All data is transitory, once a container completes its transcription it removes all record of the operation, no data is persisted.

Accessing the Image

The Speechmatics Docker images are obtainable from the Speechmatics Docker repository (jfrog.io). If you do not have a Speechmatics software repository account or have lost your details, please contact Speechmatics support support@speechmatics.com.

The latest information about the containers can be found in the solutions section of the support portal. If a support account is not available or the *Containers* section is not visible in the support portal, please contact Speechmatics support support@speechmatics.com for help.

Prior to pulling any Docker images, the following must be known:

- Speechmatics Docker URL – provided by the Speechmatics team
- Image name (which usually includes the language code of the target language, e.g. `en` for English or `de` for German)
- Image tag - which identifies the image version

Getting the Image

After gaining access to the relevant details for the Speechmatics software repository, follow the steps below to login and pull the Docker images that are required.

Software Repository Login

Ensure the *Speechmatics Docker URL* and software repository *username* and *password* are available. The endpoint being used will require Docker to be installed. For example:

```
docker login https://speechmatics-docker-example.jfrog.io
```

You will be prompted for username and password. If successful, you will see the response:

```
Login Succeeded
```

If unsuccessful, please verify your credentials and URL. If problems persist, please contact Speechmatics support.

Pulling the Image

To pull the Docker image to the local environment follow the instructions below. Each supported language pack comes as a different Docker image, so the process will need to be repeated for each required language.

Example pulling Global English (en) with the {{ book.product.version }} TAG:

```
docker pull speechmatics-docker-example.jfrog.io/rt-asr-transcriber-en:1.0.0
```

Example pulling Spanish (es) with the {{ book.product.version }} TAG:

```
docker pull speechmatics-docker-example.jfrog.io/rt-asr-transcriber-es:1.0.0
```

The image will start to download. This could take a while depending on your connection speed.

Note: Speechmatics require all customers to cache a copy of the Docker image(s) within their own environment. Please do not pull directly from the Speechmatics software repository for each deployment.

Licensing

TODO

Using the Container

Once the Docker image has been pulled into a local environment, it can be started using the Docker `run` command. More details about operating and managing the container are available in the Docker API documentation.

Here's a couple of examples of how to start the container from the command-line:

```
docker run -p 9000:9000 -p 8001:8001 speechmatics-docker-example.jfrog.io/rt-asr-transcriber-en:1.0.0
docker run -p 9000:9000 -p 8001:8001 speechmatics-docker-example.jfrog.io/rt-asr-transcriber-es:1.0.0
```

The Docker `run` options used are:

Name	Description
`--port, -p`	Expose ports on the container so that they are accessible from the host

See Docker docs for a full list of the available options.

Input Modes

The supported method for passing audio to a container is to use a Websocket. A session is setup with configuration parameters passed in using a `StartRecognition` message, and thereafter audio is sent to the container in binary chunks, with transcripts being returned in an `AddTranscript` message.

In the `AddTranscript` message individual result segments are returned, corresponding to audio segments defined by pauses (and other latency measurements).

Output

The results list in the V2 Output format are sorted by increasing `start_time`, with a supplementary rule to sort by decreasing `end_time`. Confidence precision is to 6 decimal places. See below for an example:

```
{
    "message": "AddTranscript",
    "format": "2.5",
    "metadata": {
        "transcript": "full tell radar",
        "start_time": 0.11,
        "end_time": 1.07
    },
    "results": [
        {
            "type": "word",
            "start_time": 0.11,
            "end_time": 0.40,
            "alternatives": [
                { "content": "full", "confidence": 0.7 }
            ]
```

```
        },
        {
            "type": "word",
            "start_time": 0.41,
            "end_time": 0.62,
            "alternatives": [
                { "content": "tell", "confidence": 0.6 }
            ]
        },
        {
            "type": "word",
            "start_time": 0.65,
            "end_time": 1.07,
            "alternatives": [
                { "content":"radar", "confidence": 1.0 }
            ]
        }
    ]
}
```

Health service

The container is able to expose an HTTP health service, which offers startup, liveness and readiness probes. This may be especially helpful if you are deploying the container into a Kubernetes cluster. If you are using Kubernetes, we recommend that you also refer to the Kubernetes documentation around liveness and readiness probes (https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/).

Enabling the health service

The health service is enabled by default and should run as a subprocess of the main entrypoint to the container.

Endpoints

The health service offers three endpoints:

The code examples below use the HTTPie tool

`/started`

This endpoint provides a startup probe. It can be queried using an HTTP GET request.

This probe indicates whether all services in the container have successfully started. Once it returns a successful response code, it should never return an unsuccessful response code later.

Possible responses:

- `200` if all of the services in the container have successfully started.
- `503` otherwise.

A JSON object is also returned in the body of the response, indicating the status.

Example:

```
$ http get address.of.container:8001/started
HTTP/1.0 200 OK
Content-Type: application/json
```

```
Date: Mon, 16 Mar 2020 17:28:46 GMT
Server: BaseHTTP/0.6 Python/3.5.2

{
    "started": true
}
```

`/live`

This endpoint provides a liveness probe. It can be queried using an HTTP GET request.

This probe indicates whether all services in the container are active. The services in the container send regular updates to the health service, if they don't send an update for more than 10 seconds then they will be marked as 'dead' and this endpoint will return an unsuccessful response code. For example, if the WebSocket server in the container was to crash, this endpoint should indicate that.

Possible responses:

- `200` if all of the services in the container have successfully started, and have recently sent an update to the health service.
- `503` otherwise.

A JSON object is also returned in the body of the response, indicating the status.

Example:

```
$ http get address.of.container:8001/live
HTTP/1.0 200 OK
Content-Type: application/json
Date: Mon, 16 Mar 2020 17:36:35 GMT
Server: BaseHTTP/0.6 Python/3.5.2

{
    "alive": true
}
```

`/ready`

This endpoint provides a readiness probe. It can be queried using an HTTP GET request.

This probe indicates whether the container is currently transcribing something; if the server is handling at least one audio stream then it is considered not ready.

We recommend limiting our container to one audio stream at a time, and using this probe as a scaling mechanism. That said, the container can handle multiple concurrent audio streams.

Note: The readiness check is accurate within a 2 second resolution. If you do use this probe for load balancing, be aware that bursts of traffic within that 2 second window could all be allocated to a single container since it's readiness state will not change.

Possible responses:

- `200` if the container is not currently transcribing audio.
- `503` otherwise.

A JSON object is also returned in the body of the response, indicating the status.

Example:

```
$ http get address.of.container:8001/ready
HTTP/1.0 200 OK
Content-Type: application/json
Date: Mon, 16 Mar 2020 17:37:00 GMT
Server: BaseHTTP/0.6 Python/3.5.2

{
    "ready": true
}
```

Troubleshooting

Enabling Logging

If you are seeing problems then we recommend that you open a ticket with Speechmatics support: support@speechmatics.com. Please include the logging output from the container if you do open a ticket, and ideally enable verbose logging.

Verbose logging is enabled by running the container with the environment variable `DEBUG` set to `true`.

e.g.

```
docker run -e DEBUG=true speechmatics-docker-example.jfrog.io/rt-asr-transcriber-en:1.0.0
```

Common Problems

You should ensure, when using the config object in the `StartRecognition` message, that the JSON is correctly formatted.

Real-time Container API Guide

This page specifies the Real-time API at its current state. The basic elements in the communication are the following:

- **Client** - An application connecting to the API, providing the audio and processing the transcripts received from the **Server**.
- **Server** (also called **API**) - An entry point of the API, allows external connections and provides the transcripts back.
- **Worker** - An internal speech recognizer. This is an internal entity that actually runs the heavy speech recognition.

This is a specification for Speechmatics Real-time API version 2.7

Client ↔ API endpoint

The communication is done using WebSockets, which are implemented in most of the modern web-browsers, as well as in many common programming languages (namely C++ and Python, for instance using http://autobahn.ws/).

Messages

Each message that the **Server** accepts is a stringified JSON object with the following fields:

- `message` (String): The name of the message we are sending. Any other fields depend on the value of the `message` and are described below.

The messages sent by the **Server** to a **Client** are stringified JSON objects as well.

The only exception is a binary message sent from the **Client** to the **Server** containing a chunk of audio which will be referred to as `AddAudio`.

The following values of the `message` field are supported:

StartRecognition

Initiates recognition, based on details provided in the following fields:

- `message: "StartRecognition"`
- `audio_format` (Object:AudioType): Required. Audio stream type you are going to send: see Supported audio types.
- `transcription_config` (Object:TranscriptionConfig): Required. Set up configuration values for this recognition session, see Transcription config.

A `StartRecognition` message must be sent exactly once after the WebSocket connection is opened. The client must wait for a `RecognitionStarted` message before sending any audio.

In case of success, a message with the following format is sent as a response:

- `message: "RecognitionStarted"`
- `id` (String): Required. A randomly-generated GUID which acts as an identifier for the session. e.g. "807670e9-14af-4fa2-9e8f-5d525c22156e".

In case of failure, an error message is sent, with `type` being one of the following: `invalid_model`, `invalid_audio_type`, `not_authorised`, `insufficient_funds`, `not_allowed`, `job_error`.

An example of the `StartRecognition` message:

```
{
  "message": "StartRecognition",
  "audio_format": {
    "type": "raw",
    "encoding": "pcm_f32le",
    "sample_rate": 16000
  },
  "transcription_config": {
    "language": "en",
    "output_locale": "en-US",
    "diarization": "speaker_change",
    "max_delay": 3.5,
    "max_delay_mode": "flexible",
    "enable_partials": true,
  }
}
```

Explaining Max Delay Mode

Users can specify the latency of the Real-time Speechmatics engine using the `max_delay` parameter. If a value of '5' was chosen, this would mean that transcripts would always be returned in 5 seconds from the word first being spoken. This happens even if a word is detected that may overrun that threshold. In some cases this can lead to inaccuracies in recognition and in finalised transcripts. This can be especially noticeable with key entities such as numerals, currencies, and dates.

`max_delay_mode` allows a greater flexibility in this latency only when a potential entity has been detected. Entities are common concepts such as numbers, currencies and dates, and can be seen in more detail here.

There are two potential options for `max_delay_mode`: `fixed` and `flexible`. If no option is chosen, the default is `flexible`. Where an entity is detected with `flexible`, the latency of a transcript may exceed the threshold specified in `max_delay`, however the recognition of entities will be more accurate. If a user specifies `fixed`, the transcript will be returned in segments that will never exceed the `max_delay` threshold, even if this causes inaccuracies in entity recognition.

SetRecognitionConfig

Allows the **Client** to configure the recognition session even after the initial `StartRecognition` message without restarting the connection. **This is only supported for certain parameters**.

- `message: "SetRecognitionConfig"`
- `transcription_config` (Object:TranscriptionConfig): A TranscriptionConfig object containing the new configuration for the session, see Transcription config.

The following is an example of such a configuration message:

```
{
  "message": "SetRecognitionConfig",
  "transcription_config": {
    "language": "en",
    "max_delay": 3.5,
    "enable_partials": true
  }
}
```

Note: The `language` property is a mandatory element in the `transcription_config` object; however it is not possible to change the language mid-way through the session (it will be ignored if you do). It is only possible to modify the following settings through a **SetRecognitionConfig** message after the initial `StartRecognition` message:

- `max_delay`
- `max_delay_mode`
- `enable_partials`

If you wish to alter any other parameters you must terminate the session and restart with the altered configuration. Attempting otherwise will result in an error.

The example above starts a session with the Global English model ready to consume raw PCM encoded audio with float samples at 16kHz. It also includes an `additional_vocab` list containing the names of different types of pasta. `speaker_change` diarization is enabled, and partials are enabled meaning that `AddPartialTranscript` messages will be received as well as `AddTranscript` messages. Punctuation is configured to restrict the set of punctuation marks that will be added to only commas and full stops.

AddAudio

Adds more audio data to the recognition job started on the WebSocket using `StartRecognition`. The server will only accept audio after it is initialized with a job, which is indicated by a `RecognitionStarted` message. Only one audio stream in one format is currently supported per WebSocket (and hence one recognition job). `AddAudio` is a binary message containing a chunk of audio data and no additional metadata.

AudioAdded

If the `AddAudio` message is successfully received, an AudioAdded message is sent as a response. This message confirms that the **Server** has accepted the data and will make a corresponding **Worker** process it. If the **Client**

implementation holds the data in an internal buffer to resubmit in case of an error, it can safely discard the corresponding data after this message. The following fields are present in the response:

- `message: "AudioAdded"`
- `seq_no` (Int): Required. An incrementing number which is equal to the number of audio chunks that the server has processed so far in the session. The count begins at 1 meaning that the 5th `AddAudio` message sent by the client, for example, should be answered by an `AudioAdded` message with `seq_no` equal to 5.

Possible errors:

- `data_error`, `job_error`, `buffer_error`

When sending audio faster than real time (for instance when sending files), make sure you don't send too much audio ahead of time. For large files, this causes the audio to fill out networking buffers, which might lead to disconnects due to WebSocket ping/pong timeout. Use AudioAdded messages to keep track what messages are processed by the engine, and don't send more than 10s of audio data or 500 individual AddAudio messages ahead of time (whichever is lower).

Implementation details

Under special circumstances, such as when the client is sending the audio data faster than real time, the **Server** might read the data slower than the **Client** is sending it. The **Server** will not read the binary `AddAudio` message if it is larger than the internal audio buffer on the **Server**. Note that for each **Worker**, there is a separate buffer. In that case, the server will read any messages coming in on the WebSocket, until enough space is made in the buffer by passing the data to a corresponding **Worker**. The **Client** will only receive the corresponding AudioAdded response message once the binary data is read. The WebSocket might eventually fill all the TCP buffers on the way, causing a corresponding WebSocket to fail to write and close the connection with prejudice. The **Client** can use the bufferedAmount attribute of the WebSocket to prevent this.

AddTranscript

This message is sent from the **Server** to the **Client**, when the **Worker** has provided the **Server** with a segment of transcription output. It contains the transcript of a part of the audio the **Client** has sent using `AddAudio` - the **final transcript**. These messages are also referred to as **finals**. Each message corresponds to the audio since the last `AddTranscript` message. The transcript is final - any further `AddTranscript` or `AddPartialTranscript` messages will only correspond to the newly processed audio. An `AddTranscript` message is sent when we reach an endpoint (end of a sentence or a phrase in the audio), or after 10s if we haven't reached such an event. This timeout can be further configured by setting `transcription_config.max_delay` in the `StartRecognition` message.

- `message: "AddTranscript"`
- `metadata` (Object): Required.
  - `start_time` (Number): Required. An approximate time of occurrence (in seconds) of the audio corresponding to the beginning of the first word in the segment.
  - `end_time` (Number): Required. An approximate time of occurrence (in seconds) of the audio corresponding to the ending of the final word in the segment.
  - `transcript` (String): Required. The entire transcript contained in the segment in text format. Providing the entire transcript here is designed for ease of consumption; we have taken care of all the necessary formatting required to concatenate the transcription results into a block of text. This transcript lacks the detailed information however which is contained in the `results` field of the message - such as the timings and confidences for each word.
- `results` (List:Object):
  - `type` (String): Required. One of 'word', 'entity', 'punctuation' or 'speaker_change'. 'word' results represent a single word. 'punctuation' results represent a single punctuation symbol. 'word' and 'punctuation' results will both have one or more `alternatives` representing the possible

alternatives we think the word or punctuation symbol could be. 'entity' is only a possible type if `enable_entities` is set to `true` and indicates a formatted entity. 'speaker_change' results however will have an empty `alternatives` field. 'speaker_change' results will only occur when using speaker_change diarization.

- `start_time` (Number): Required. The start time of the result **relative to** the start_time of the whole segment as described in `metadata`.
- `end_time` (Number): Required. The end time of the result **relative to** the start_time of the segment in the message as described in `metadata`. Note that punctuation symbols and speaker_change results are considered to be zero-duration and thus for those results `start_time` is equal to `end_time`.
- `is_eos` (Boolean): Optional. Only present for 'punctuation' results. This indicates whether or not the punctuation mark is considered an end-of-sentence symbol. For example full-stops are an end-of-sentence symbol in English, whereas commas are not. Other languages, such as Japanese, may use different end-of-sentence symbols.
- `alternatives` (List:Object): Optional. For 'word' and 'punctuation' results this contains a list of possible alternative options for the word/symbol.
  - `content` (String): Required. A word or punctuation mark. When `enable_entities` is requested this can be multiple words with spaces, for example "17th of January 2022".
  - `confidence` (Number): Required. A confidence score assigned to the alternative. Ranges from 0.0 (least confident) to 1.0 (most confident).
  - `display` (Object): Optional. Information about how the word/symbol should be displayed.
    - `direction` (String): Required. Either 'ltr' for words that should be displayed left-to-right, or 'rtl' vice versa.
  - `language` (String): Optional. The language that the alternative word is assumed to be spoken in. Currently this will always be equal to the language that was requested in the initial `StartRecognition` message.
  - `tags` (array): Optional. Only `[disfluency]` and `[profanity]` are displayed. This is a set list of profanities and disfluencies respecitvely that cannot be altered by the end user. `[disfluency]` is only present in English, and `[profanity]` is present in English, Spanish, and Italian.
- `entity_class` (String): Optional. If `enable_entities` is requested in the startTranscriptionConfig request, and an entity is detected, `entity_class` will represent the type of entity the word(s) have been formatted as.
- `spoken_form` (List:Object): Optional. If `enable_entities` is requested in the startTranscriptionConfig request, and an entity is detected, this is a list of words without formatting applied. This follows the `results` list API formatting.
- `written_form` (List:Object): Optional. If `enable_entities` is requested in the startTranscriptionConfig request, and an entity is detected, this is a list of formatted words that matches the entity `content` but with individual estimated timing and confidences. This follows the `results` list API formatting.

AddPartialTranscript

A partial-transcript message. The structure is the same as `AddTranscript`. A partial transcript is a transcript that can be changed and expanded by a future `AddTranscript` or `AddPartialTranscript` message and corresponds to the part of audio since the last `AddTranscript` message. For `AddPartialTranscript` messages the `confidence` field for `alternatives` has no meaning and will always be equal to 0.

Partials will only be sent if `transcription_config.enable_partials` is set to `true` in the `StartRecognition` message.

EndOfStream

This message is sent from the Client to the API to announce that it has finished sending all the audio that it intended to send. No more `AddAudio` message are accepted after this message. The Server will finish processing the audio it has received already and then send an EndOfTranscript message. This message is usually sent at the end of file or when the microphone input is stopped.

- `message: "EndOfStream"`
- `last_seq_no` (Int): Required. The total number of audio chunks sent (in the `AddAudio` messages).

EndOfTranscript

Sent from the API to the Client when the API has finished all the audio, as marked with the `EndOfStream` message. The API sends this only after it sends all the corresponding `AddTranscript` messages first. Upon receiving this message the Client can safely disconnect immediately because there will be no more messages coming from the API.

Supported audio types

An `AudioType` object always has one mandatory field `type`, and potentially more mandatory fields based on the value of `type`. The following types are supported:

`type: "raw"`

Raw audio samples, described by the following additional mandatory fields:

- `encoding` (String): Encoding used to store individual audio samples. Currently supported values:
  - `pcm_f32le` - Corresponds to 32 bit float PCM used in the WAV audio format, little-endian architecture. 4 bytes per sample.
  - `pcm_s16le` - Corresponds to 16 bit signed integer PCM used in the WAV audio format, little-endian architecture. 2 bytes per sample.
  - `mulaw` - Corresponds to 8 bit µ-law (mu-law) encoding. 1 byte per sample.
- `sample_rate` (Int): Sample rate of the audio

Please ensure when sending raw audio samples in real-time that the samples are undivided. For example, if you are sending raw audio via `pcm_f32le`, the sample should always contain 4 bytes. Here, if a sample did not contain 4 bytes, and then an EndOfStream message were sent, this would then cause an error. Required byte sizes per sample for each type of raw audio are listed above.

`type: "file"`

Any audio/video format supported by GStreamer. The `AddAudio` messages have to provide all the file contents, including any headers. The file is usually not accepted all at once, but segmented into reasonably sized messages.

Example `audio_format` field value: `audio_format: {type: "raw", encoding: "pcm_s16le", sample_rate: 44100}`

Transcription config

A `TranscriptionConfig` object specifies various configuration values for the recognition engine. All the values are optional, using default values when not provided.

- `language` (String): Required. Language model to process the audio input, normally specified as an ISO language code e.g. 'en'.
- `additional_vocab` (List:AdditionalWord): Optional. Configure **additional words**. See Additional words. Default is an empty list. You should be aware that there is a performance penalty (latency degradation and memory increase) from using `additional_vocab`, especially if you intend to load in a large word list.

When initialising a session that uses `additional_vocab` in the config you should expect a delay of up to 15 seconds, and an additional 800MB to 1700MB of memory (depending on the size of the list).

- `diarization` (String): Optional. The speaker diarization method to apply to the audio. The default is "none" indicating that no diarization will be performed. An alternative option is "speaker_change" diarization in which the ASR system will attempt to detect any changes in speaker. Speaker changes are indicated in the results using an object with a `type` set to `speaker_change`. Speaker change is a beta feature.
- `enable_partials` (Boolean): Optional. Whether or not to send partials (i.e. `AddPartialTranscript` messages) as well as finals (i.e. `AddTranscript` messages). The default is `false`.
- `max_delay` (Number): Optional. Maximum delay in seconds between receiving input audio and returning partial transcription results. The default is 10. The minimum and maximum values are 2 and 20.
- `output_locale` (String): Optional. Configure **output locale**. See Output locale. Default is an empty string.
- `punctuation_overrides` (Object:PunctuationOverrides): Optional. Options for controlling punctuation in the output transcripts. See Punctuation overrides.
- `speaker_change_sensitivity` (Number): Optional.: Controls how responsive the system is for potential speaker changes. The value ranges between zero and one. High value indicates high sensitivity, i.e. prefer to indicate a speaker change if in doubt. The default is 0.4. This setting is only applicable when using `"diarization": "speaker_change"`.
- `operating_point` (String): Optional. Which model within the language pack you wish to use for transcription with a choice between `standard` and `enhanced`. See API How-to Guide for more details
- `enable_entities` (Boolean): Optional. Whether a user wishes for entities to be identified with additional spoken and written word format. Supported values `true` or `false`. The default is `false`.

Requesting an enhanced model

Speechmatics supports two different models within each language pack; a standard or an enhanced model. The standard model is the faster of the two, whilst the enhanced model provides a higher accuracy, but a slower turnaround time.

The enhanced model is a premium model. Please contact your account manager or Speechmatics if you would like access to this feature.

An example of requesting the enhanced model is below

```
{
  "message": "StartRecognition",
  "audio_format": {
    "type": "raw",
    "encoding": "pcm_f32le",
    "sample_rate": 16000
  },
{
  "transcription_config": {
    "language": "en",
    "operating_point": "enhanced"
  }
}
```

Please note: `standard`, as well as being the default option, can also be explicitly requested with the `operating_point` parameter.

Additional words

Additional words expand the standard recognition dictionary with a list of words or phrases called **additional words**. An **additional word** can also be a phrase, as long as individual words in the phrase are separated by spaces. This is the **custom dictionary** supported in other Speechmatics products. A pronunciation of those words is generated automatically or based on a provided `sounds_like` field. This is intended for adding new words and phrases, such as domain-specific terms or proper names. Better results for domain-specific words that contain common words can be achieved by using phrases rather than individual words (such as `action plan`).

`AdditionalWord` is either a `String` (the **additional word**) or an `Object`. The object form was introduced in 0.7.0. The object form has the following fields:

- `content` (String): The **additional word**.
- `sounds_like` (List:String): A list of words with similar pronunciation. Each word in this list is used as one alternative pronunciation for the additional word. These don't have to be real words - only their pronunciation matters. This list must not be empty. Words in the list must not contain whitespace characters. When `sounds_like` is used, the pronunciation automatically obtained from the `content` field is not used.

The `String` form `"word"` corresponds with the following `Object` form: `{"content": "word", "sounds_like": ["word"]}`.

Full example of `additional_vocab`:

```
    "additional_vocab": [
        "speechmatics",
        {"content": "gnocchi", "sounds_like": ["nyohki", "nokey", "nochi"]},
        {"content": "CEO", "sounds_like": ["seeoh"]},
        "financial crisis"
    ]
```

To clarify, the following ways of adding the word `speechmatics` are equivalent with all using the default pronunciation:

1. `"additional_vocab": ["speechmatics"]`
2. `"additional_vocab": [{"content": "speechmatics"}]`
3. `"additional_vocab": [{"content": "speechmatics", "sounds_like": ["speechmatics"]}]`

Output locale

Change the spellings of the transcription according to the output locale language code. If the selected language pack supports a different output locale, this config value can be used to provide spelling for the transcription in one of these locales. For example, the English language pack currently supports locales: `en-GB`, `en-US` and `en-AU`. The default value for `output_locale` is an empty string that means the transcription will use its default configuration (without spellings being altered in the transcription).

The following locales are supported for Chinese Mandarin. The default is simplified Mandarin.

- Simplified Mandarin (cmn-Hans)
- Traditional Mandarin (cmn-Hant)

Punctuation overrides

This object contains settings for configuring punctuation in the transcription output.

- `permitted_marks` (List:String) Optional. The punctuation marks which the client is prepared to accept in transcription output, or the special value 'all' (the default). Unsupported marks are ignored. This value is used to guide the transcription process.

- `sensitivity` (Number) Optional. Ranges between zero and one. Higher values will produce more punctuation. The default is 0.5.

Error messages

Error messages have the following fields:

- `message: "Error"`
- `code` (Int): Optional. A numerical code for the error. See below. TODO: This is not yet finalised.
- `type` (String): Required. A code for the error message. See the list of possible errors below.
- `reason` (String): Required. A human-readable reason for the error message.

Error types

- **`type: "invalid_message"`**
  - The message received was not understood.
- **`type: "invalid_model"`**
  - Unable to use the model for the recognition. This can happen if the language is not supported at all, or is not available for the user.
- **`type: "invalid_config"`**
  - The config received contains some wrong/unsupported fields.
- **`type: "invalid_audio_type"`**
  - Audio type is not supported, is deprecated, or the audio_type is malformed.
- **`type: "invalid_output_format"`**
  - Output format is not supported, is deprecated, or the output_format is malformed.
- **`type: "not_authorised"`**
  - User was not recognised, or the API key provided is not valid.
- **`type: "insufficient_funds"`**
  - User doesn't have enough credits or any other reason preventing the user to be charged for the job properly.
- **`type: "not_allowed"`**
  - User is not allowed to use this message (is not allowed to perform the action the message would invoke).
- **`type: "job_error"`**
  - Unable to do any work on this job, the **Worker** might have timed out etc.
- **`type: "data_error"`**
  - Unable to accept the data specified - usually because there is too much data being sent at once
- **`type: "buffer_error"`**
  - Unable to fit the data in a corresponding buffer. This can happen for clients sending the input data faster then real-time.
- **`type: "protocol_error"`**
  - Message received was syntactically correct, but could not be accepted due to protocol limitations. This is usually caused by messages sent in the wrong order.
- **`type: "unknown_error"`**
  - An error that did not fit any of the types above.

Note that `invalid_message`, `protocol_error` and `unknown_error` can be triggered as a response to any type of messages.

The transcription is terminated and the connection is closed after any error.

Warning messages

Warning messages have the following fields:

- `message: "Warning"`
- `code` (Int): Optional. A numerical code for the warning. See below. TODO: This is not yet finalised.
- `type` (String): Required. A code for the warning message. See the list of possible warnings below.
- `reason` (String): Required. A human-readable reason for the warning message.

Warning types

- **`type: "duration_limit_exceeded"`**
  - The maximum allowed duration of a single utterance to process has been exceeded. Any AddAudio messages received that exceed this limit are confirmed with AudioAdded, but are ignored by the transcription engine. Exceeding the limit triggers the same mechanism as receiving an `EndOfStream` message, so the Server will eventually send an `EndOfTranscript` message and suspend.
  - It has the following extra field:
    - `duration_limit` (Number): The limit that was exceeded (in seconds).

Info messages

Info messages denote additional information sent form the **Server** to the **Client**. Those are similar to `Error` and `Warning` messages in syntax, but don't actually denote any problem. The **Client** can safely ignore these messages or use them for additional client-side logging.

- `message: "Info"`
- `code` (Int): Optional. A numerical code for the informational message. See below. TODO: This is not yet finalised.
- `type` (String): Required. A code for the info message. See the list of possible info messages below.
- `reason` (String): Required. A human-readable reason for the informational message.

Info message types

- **`type: "recognition_quality"`**
  - Informs the client what particular quality-based model is used to handle the recognition.
  - It has the following extra field:
    - `quality` (String): Quality-based model name. It is one of `"telephony"`, `"broadcast"`. The model is selected automatically, for high-quality audio (12kHz+) the broadcast model is used, for lower quality audio the telephony model is used.
- ** `type: "model_redirect"`
  - Informs the client that a deprecated language code has been specified, and will be handled with a different model. For example, if the `model` parameter is set to one of en-US, en-GB, or en-AU, then the request may be internally redirected to the Global English model (en).
- ** `type: "deprecated"`
  - Informs about using a feature that is going to be removed in a future release.

Example communication

The communication consists of 3 stages - initialization, transcription and a disconnect handshake.

On **initialization**, the `StartRecognition` message is sent from the Client to the API and the Client must block and wait until it receives a `RecognitionStarted` message.

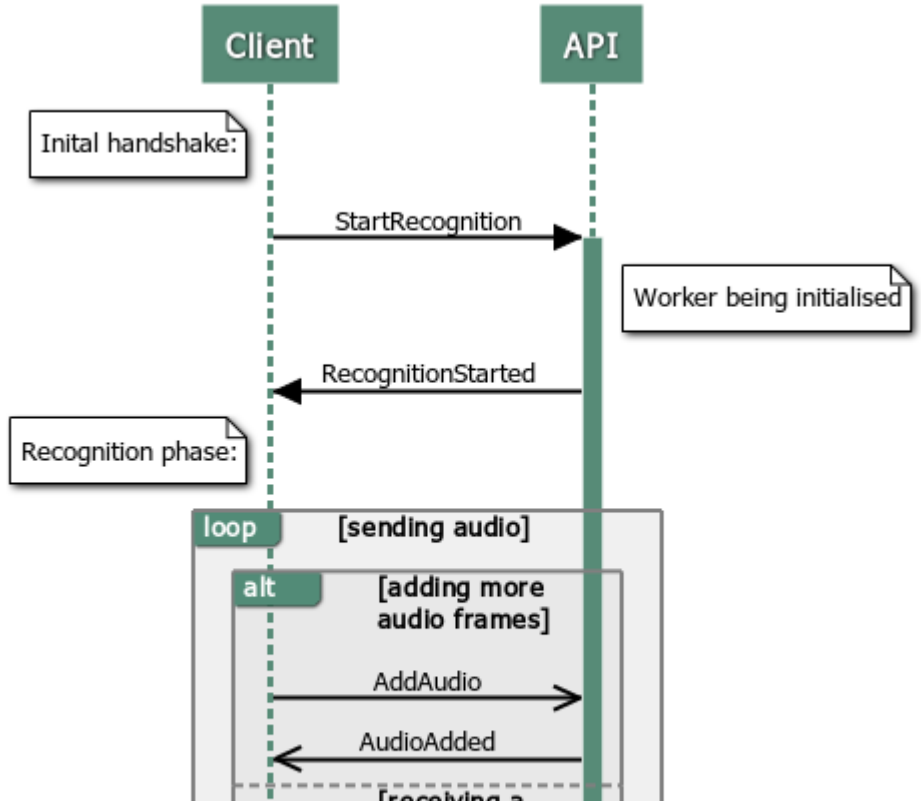
Afterwards, the **transcription** stage happens. The client keeps sending `AddAudio` messages. The API asynchronously replies with `AudioAdded` messages. The API also asynchronously sends `AddPartialTranscript` and `AddTranscript` messages.

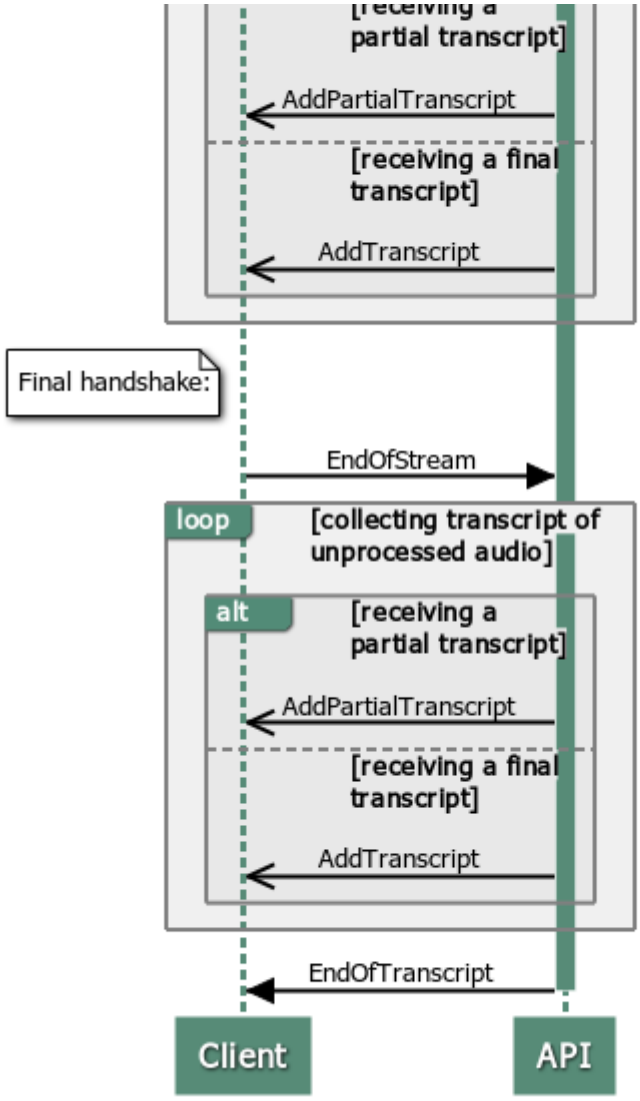
Once the client doesn't want to send any more audio, the **disconnect handshake** is performed. The Client sends an `EndOfStream` message as it's last message. No more messages are handled by the API afterwards. The API processes whatever audio it has buffered at that point and sends all the `AddTranscript` and `AddPartialTranscript` messages accordingly. Once the API processes all the buffered audio, it sends an `EndOfTranscript` message and the Client can then safely disconnect.

Note: In the example below, `->` denotes a message sent by the Client to the API, `<-` denotes a message send by the API to the Client. Any comments are denoted `"[like this]"`.

```
-> {"message": "StartRecognition", "audio_format": {"type": "file"},
    "transcription_config": {"language": "en", "enable_partials": true}}

 <- {"message": "RecognitionStarted", "id": "807670e9-14af-4fa2-9e8f-5d525c22156e"}

->  "[binary message - AddAudio 1]"
->  "[binary message - AddAudio 2]"

 <- {"message": "AudioAdded", "seq_no": 1}
 <- {"message": "Info", "type": "recognition_quality", "quality": "broadcast", "reason":
"Running recognition using a broadcast model quality."}
 <- {"message": "AudioAdded", "seq_no": 2}

->  "[binary message - AddAudio 3]"

 <- {"message": "AudioAdded", "seq_no": 3}

"[asynchronously received transcripts:]"

 <- {"message": "AddPartialTranscript", "metadata": {"start_time": 0.0, "end_time":
0.5399999618530273, "transcript": "One"},
     "results": [{"alternatives": [{"confidence": 0.0, "content": "One"}],
                  "start_time": 0.47999998927116394, "end_time": 0.5399999618530273,
"type": "word"}
                ]}
 <- {"message": "AddPartialTranscript", "metadata": {"start_time": 0.0, "end_time":
0.7498992613545260, "transcript": "One to"},
     "results": [{"alternatives": [{"confidence": 0.0, "content": "One"}],
                  "start_time": 0.47999998927116394, "end_time": 0.5399999618530273,
"type": "word"},
                 {"alternatives": [{"confidence": 0.0, "content": "to"}],
                  "start_time": 0.6091238623430891, "end_time": 0.7498992613545260, "type":
"word"}
                ]}
 <- {"message": "AddPartialTranscript", "metadata": {"start_time": 0.0, "end_time":
0.9488123643240011, "transcript": "One to three"},
     "results": [{"alternatives": [{"confidence": 0.0, "content": "One"}],
                  "start_time": 0.47999998927116394, "end_time": 0.5399999618530273,
```

```
"type": "word"},
                {"alternatives": [{"confidence": 0.0, "content": "to"}],
                 "start_time": 0.6091238623430891, "end_time": 0.7498992613545260, "type":
"word"}
                {"alternatives": [{"confidence": 0.0, "content": "three"}],
                 "start_time": 0.8022338627780892, "end_time": 0.9488123643240011, "type":
"word"}
               ]}
 <- {"message": "AddTranscript", "metadata": {"start_time": 0.0, "end_time":
0.9488123643240011, "transcript": "One two three."},
     "results": [{"alternatives": [{"confidence": 1.0, "content": "One"}],
                  "start_time": 0.47999998927116394, "end_time": 0.5399999618530273,
"type": "word"},
                 {"alternatives": [{"confidence": 1.0, "content": "to"}],
                  "start_time": 0.6091238623430891, "end_time": 0.7498992613545260, "type":
"word"}
                 {"alternatives": [{"confidence": 0.96, "content": "three"}],
                  "start_time": 0.8022338627780892, "end_time": 0.9488123643240011, "type":
"word"}
                 {"alternatives": [{"confidence": 1.0, "content": "."}],
                  "start_time": 0.9488123643240011, "end_time": 0.9488123643240011, "type":
"punctuation", "is_eos": true}
                ]}

"[closing handshake]"

->  {"message":"EndOfStream","last_seq_no":3}

 <- {"message": "EndOfTranscript"}
```





Examples how to use the V2 API

The V2 WebSocket Speech API aligns with other Speechmatics platforms such as the Batch Virtual Appliance and Speechmatics Cloud Offering.

WebSocket URI

To use the V2 API you use the '/v2' endpoint for the URI, for example:

```
ws://rt-asr.example.com:9000/v2
```

If you are using the Real-time Container then you will need to use the ws:// scheme, for example: `ws://rt-asr.example.com:9000/v2`. If you need to access the Real-time Container over a secure WebSocket connection from you client, then you'll need to consider an SSL offload from a load-balancer or similar.

Session Configuration

The V2 API is configured by sending a `StartRecognition` message initially when the WebSocket connection begins. We have designed the format of this message to be very similar to the `config.json` object that has been used for a while now with the Speechmatics batch mode platforms (Batch Virtual Appliance, Batch Container and Cloud Offering). The `transcription_config` section of the message should be almost identical between the two modes. There are some minor differences (for example batch features a different set of diarization options, and real-time features some settings which don't apply to batch such as `max_delay`).

TranscriptionConfig

A `transcription_config` structure is used to specify various configuration values for the recognition engine when the `StartRecognition` message is sent to the server. All values apart from `language` are optional. Here's an example of calling the `StartRecognition` message with this structure:

```
{
   "message": "StartRecognition",
   "transcription_config": {
      "language": "en"
   },
   "audio_format": {
      "type": "raw",
      "encoding": "pcm_f32le",
      "sample_rate": 16000
   }
}
```

AddAudio

Once the websocket session is setup and you've successfully called `StartRecognition` you'll receive a `RecognitionStarted` message from server. You can then just to send the binary audio chunks, which we refer to as `AddAudio` messages.

You would replace this in the V2 API with much simpler code:

```
// NEW V2 EXAMPLE
function addAudio(audioData) {
    ws.send(audioData);
    seqNoIn++;
}
```

We recommend that you do not send more than 10 seconds of audio data or 500 individual AddAudio messages ahead of time.

Final and Partial Transcripts

The `AddTranscript` and `AddPartialTranscript` messages from the server output a JSON format which aligns with the JSON output format used by other Speechmatics products. There is a now a `results` list which contains the transcribed words and punctuation marks along with timings and confidence scores. Here's an example of a final transcript output:

```
{
   "message":"AddTranscript",
   "results":[
      {
         "start_time":0.11670026928186417,
         "end_time":0.4049381613731384,
```

```
        "alternatives":[
           {
              "content":"gale",
              "confidence":0.7034434080123901
           }
        ],
        "type":"word"
     },
     {
        "start_time":0.410246878862381,
        "end_time":0.6299981474876404,
        "alternatives":[
           {
              "content":"eight",
              "confidence":0.670033872127533
           }
        ],
        "type":"word"
     },
     {
        "start_time":0.6599999666213989,
        "end_time":1.0799999237060547,
        "alternatives":[
           {
              "content":"becoming",
              "confidence":1.0
           }
        ],
        "type":"word"
     },
     {
        "start_time":1.0799999237060547,
        "end_time":1.6154180765151978,
        "alternatives":[
           {
              "content":"cyclonic",
              "confidence":1.0
           }
        ],
        "type":"word"
     },
     {
        "start_time":1.6154180765151978,
        "is_eos":true,
        "end_time":1.6154180765151978,
        "alternatives":[
           {
              "content":".",
              "confidence":1.0
           }
        ],
        "type":"punctuation"
     }
  ],
```

```
   "metadata":{
      "transcript":"gale eight becoming cyclonic.",
      "start_time":190.65994262695312,
      "end_time":194.46994256973267
   },
   "format":"2.7"
}
```

You can use the `metadata.transcript` property to get the complete final transcript as a chunk of plain text. The `format` property describes the exact version of the transcription output format, which is currently 2.7. This may change in future releases if the output format is updated.

Requesting an enhanced model

Speechmatics supports two different models within each language pack; a standard or an enhanced model. The standard model is the faster of the two, whilst the enhanced model provides a higher accuracy, but a slower turnaround time.

The enhanced model is a premium model. Please contact your account manager or Speechmatics if you would like access to this feature.

An example of requesting the enhanced model is below

```
{
  "message": "StartRecognition",
  "audio_format": {
    "type": "raw",
    "encoding": "pcm_f32le",
    "sample_rate": 16000
  },
{
  "transcription_config": {
    "language": "en",
    "operating_point": "enhanced"
  }
}
```

Please note: `standard`, as well as being the default option, can also be explicitly requested with the `operating_point` parameter.

Advanced Punctuation

Some language models (Arabic, Danish, Dutch, English, French, German, Malay, Spanish, Swedish and Turkish currently) support advanced punctuation. This uses machine learning techniques to add in more naturalistic punctuation, improving the readability of your transcripts. As well as putting punctuation marks in more naturalistic positions in the output, additional punctuation marks such as commas (,) exclamation marks (!) and question marks (?) will also appear.

There is no need to explicitly enable this in the configuration; languages that support advanced punctuation will automatically output these marks. If you do not want to see these punctuation marks in the output, then you can explicitly control this through the `punctuation_overrides` setting within the `transcription_config` object, for example:

```
    "transcription_config": {
        "language": "en",
        "punctuation_overrides": {
```

```
            "permitted_marks": [ "." ]
        }
    }
```

Note that changing the punctuation setting from the default can take a couple of seconds, which means if the user is using non-default neural punctuation sensitivity, after they send the `StartRecognition` message, there will be a slight delay (2-3 seconds) before the `RecognitionStarted` message is sent back.

The JSON output places punctuation marks in the results list marked with a `type` of `"punctuation"`. So you can also filter on the output if you want to modify or remove punctuation.

Example Usage

This section provides some client code samples that show simple usage of the V2 WebSockets Speech API. It shows how you can test your Real-Time Appliance or Container using a minimal WebSocket client.

JavaScript

The basic usage of the WebSockets interface from a JavaScript client is shown in this section. In the first instance you setup the connection to the server and define the various event handlers that are required:

```
var ws = new WebSocket('ws://rtc:9000/v2');
ws.binaryType = "arraybuffer";
ws.onopen = function(event) { onOpen(event) };
ws.onmessage = function(event) { onMessage(event) };
ws.onclose = function(event) { onClose(event) };
ws.onerror = function(event) { onError(event) };
```

Change the hostname from the above example to match the IP address or hostname of your Real-Time Appliance or Container. The port used is 9000 and you need to make sure that you add '/v2' to the WebSocket URI. Note that the Real-time Container only supports WebSocket (ws) protocol. You should also ensure that the binaryType property of the WebSocket object is set to `"arraybuffer"`.

In the `onopen` handler you initiate the session by sending the **StartRecognition** message to the server, for example:

```
function onOpen(evt) {
    var msg = {
        "message": "StartRecognition",
        "transcription_config": {
            "language": "en",
            "output_locale": "en-GB"
        },
        "audio_format": {
            "type": "raw",
            "encoding": "pcm_s16le",
            "sample_rate": 16000
        }
    };

    ws.send(JSON.stringify(msg));
}
```

An `onmessage` handler is where you will respond to the *server-initiated messages* sent by the appliance or container, and decide how to handle them. Typically, this involves implementing functions to display or process data that you get back from the server.

```
function onMessage(evt) {
    var objMsg = JSON.parse(evt.data);

    switch (objMsg.message) {
        case "RecognitionStarted":
            recognitionStarted(objMsg); // TODO
            break;

        case "AudioAdded":
            audioAdded(objMsg);  // TODO
            break;

        case "AddPartialTranscript":
        case "AddTranscript":
            transcriptOutput(objMsg);  // TODO
            break;

        case "EndOfTranscript":
            endTranscript();  // TODO
            break;

        case "Info":
        case "Warning":
        case "Error":
            showMessage(objMsg);  // TODO
            break;

        default:
            console.log("UNKNOWN MESSAGE: " + objMsg.message);
    }
}
```

Once the WebSocket is initialized, the **StartRecognition** message is sent to the appliance or container to setup the audio input. It is then a matter of sending audio data periodically using the **AddAudio** message.

Your **AddAudio** message will take audio from a source (for example microphone input, or an audio stream) and pass it to the Real-Time Appliance or Container.

```
// Send audio data to the API using the AddData message.
function addAudio(audioData) {
   ws.send(audioData);
   seqNoIn++;
}
```

In this example we use a counter `seqNoIn` to keep track of the AddAudio messages we've sent.

A set of server-initiated transcript messages are triggered to indicate the availability of transcribed text:

- `AddTranscript`
- `AddPartialTranscript`

See above for changes to the JSON output schema in the V2 API. For full details of the output schema refer to the AddTranscript section in the API reference.

Finally, the client should send an **EndOfStream** message and close the WebSocket when it terminates. This should be done in order to release resources on the appliance or container and allow other clients to connect and use resources.

The Mozilla developer network provides a useful reference to the WebSocket API.

Python

Standalone Real-Time Container Usage

If you are using the Real-Time Container, you can use a Python library called `speechmatics-python`. This library is available on Github here. You can also use this library for the Real-Time Virtual Appliance.

The `speechmatics-python` library can be incorporated into your own applications, used as a reference for your own client library, or called directly from the command line (CLI) like this (to pass a test audio file to the appliance or container):

```
speechmatics transcribe --url ws://rtc:9000/v2 --lang en --ssl-mode none test.mp3
```

Note that configuration options are specified on the command-line as parameters, with a '_' character in the configuration option being replaced by a '-'. The CLI option accepts an audio stream on standard input, meaning that you can stream in a live microphone feed. To get help on the CLI use the following command:

```
speechmatics transcribe --help
```

The library depends on Python 3.7 or above, since it makes use of some of the newer `asyncio` features introduced with Python 3.7.

Formatting Common Entities

Overview

Entities are commonly recognisable classes of information that appear in languages, for example numbers and dates. Formatting these entities is commonly referred to as Inverse Text Normalisation (ITN). Using ITN, Speechmatics will output entities in a predictable, consistent written form, reducing post-processing work required aiming to make the transcript more readable.

The language pack will use these formatted entities by default in the transcription. Additional metadata about these entities can be requested via the API including the spoken words without formatting and the entity class that was used to format it.

Supported Languages

Entities are supported in the following languages:

- Cantonese
- Chinese Mandarin (Simplified and Traditional)
- English
- French
- German
- Hindi
- Italian

- Japanese
- Portuguese
- Russian
- Spanish

Using the enable_entities parameter

Speechmatics now includes an `enable_entities` parameter. This can be requested via the API. By default this is `false`.

Changing `enable_entities` to `true` will enable a richer set of metadata in the JSON output only. Customers can choose between the default written form, spoken form, or a mixture, for their own workflows.

The changes are as following:

- A new `type` - `entity` in the JSON output in addition to `word` and `punctuation`. For example: "1.99" would have a `type` of `entity` and a corresponding `entity_class` of `decimal`
- The `entity` will contain the formatted text in the `content` section, like other words and punctuation
  - The `content` can include spaces, non-breaking spaces, and symbols (e.g. $/£/%)
- A new output element `entity`, `entity_class` has been introduced. This provides more detail about how the entity has been formatted. A full list of entity classes is provided below.
- The start and end time of the entity will span all the words that make up that entity
- The entity also contains two ways that the content will be output:
  - `spoken_form` - Each individual `word` within the entity, written out in words as it was spoken. Each individual word has its own start time, end time, and confidence score. For example: "one", "million", "dollars"
  - `written_form` - The same output as within `entity` content, with a type of `word` instead. If there are spaces in the content it will be split into individual words. For example: "$1", "million"

Configuration example

Please see an example configuration file that would request entities:

```
{
  "message": "StartRecognition",
  "transcription_config": {
        "language": "en",
        "enable_entities": true
    }
}
```

Different entity classes

The following `entity_classes` can be returned. Entity classes indicate how the numerals are formatted. In some cases, the choice of class can be contextual and the class may not be what was expected (for example "2001" may be a "cardinal" instead of "date"). The number of `entity_classes` may grow or shrink in the future.

N.B. Please note existing behaviour for English where numbers from zero to 10 (excluding where they are output as a decimal/money/percentage) are output as **words** is unchanged.

Entity Class	Formatting Behaviour	Spoken Word Form Example	Written Form Example
alphanum	A series of three or more	triple seven five four	77754

	alphanumerics, where an alphanumeric is a digit less than 10, a character or symbol		
cardinal	Any number greater than ten is converted to numbers. Numbers ten or below remain as words. Includes negative numbers	nineteen	19
credit card	A long series of spoken digits less than 10 are converted to numbers. Support for common credit cards	one one one one two two two two three three three three four four four four	1111222233334444
date	Day, month and year, or a year on its own. Any words spoken in the date are maintained (including "the" and "of")	fifteenth of January twenty twenty two	15th of January 2022
decimal	A series of numbers divided by a separator	eighteen point one two	18.12
fraction	Small fractions are kept as words ("half"), complex fractions are converted to numbers separated by "/"	three sixteenths	3/16
money	Currency words are converted to symbols before or after the number (depending on the language)	twenty dollars	$20
ordinal	Ordinals greater than 10 are output as numbers	forty second	42nd
percentage	Numbers with a per cent have the per cent converted to a % symbol	duecento percento	200%
span	A range expressed as "x to y" where x and y correspond to another entity class	one hundred to two hundred million pounds	100 to £200 million
time	Times are converted to numbers	eleven forty a m	11:40 a.m.
word	Entities that do not match a specific class	hundreds	hundreds

Output locale styling

Each language has a specific style applied to it for thousands, decimals and where the symbol is positioned for money or percentages.

For example

- English contains commas as separators for numbers above 9999 (example: "20,000"), the money symbol at the start (example: "$10") and full stops for decimals (example: "10.5")
- German contains full stops as separators for numbers above 9999 (example: "20.000"), the money symbol comes after with a non-breaking space (example: "10 $") and commas for decimals (example: "10,5")
- French contains non-breaking spaces as separators for numbers above 9999 (example: "20 000"), the money symbol comes after with a non-breaking space (example: "10 $") and commas for decimals (example: "10,5")

Example output

Here is an example of a transcript requested with `enable_entities` set to `true`:

- An `entity` that is "17th of January 2022", including spaces
  - The start and end times span the entire entity
  - An `entity_class` of `date`
  - The `spoken_form` is split into the following individual words: "seventeenth", "of", "January", "twenty", "twenty", "two". Each word has its own start and end time
  - the `written_form` split into the following individual words: "17th", "of", "January", "2022". Each word has its own start and end time


```
[{
 "message": "AddTranscript",
 "format": 2.7,
 "results": [{
     "alternatives": [{
         "confidence": 1,
         "content": "17th of January 2022",
         "language": "en"
     }],
     "end_time": 3.0899999141693115,
     "entity_class": "date",
     "spoken_form": [{
             "alternatives": [{
                 "confidence": 1,
                 "content": "Seventeenth",
                 "language": "en"
             }],
             "end_time": 1.3799999952316284,
             "start_time": 0.8399999737739563,
             "type": "word"
         },
         {
             "alternatives": [{
                 "confidence": 1,
                 "content": "of",
                 "language": "en"
             }],
             "end_time": 1.4399999380111694,
             "start_time": 1.3799999952316284,
             "type": "word"
         },
         {
             "alternatives": [{
                 "confidence": 1,
                 "content": "January",
                 "language": "en"
             }],
             "end_time": 1.9199999570846558,
             "start_time": 1.4399999380111694,
             "type": "word"
         },
```

```
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "twenty",
                    "language": "en"
                }],
                "end_time": 2.25,
                "start_time": 1.9199999570846558,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "twenty",
                    "language": "en"
                }],
                "end_time": 2.549999952316284,
                "start_time": 2.25,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 0.9504331946372986,
                    "content": "two",
                    "language": "en"
                }],
                "end_time": 3.0899999141693115,
                "start_time": 2.549999952316284,
                "type": "word"
            }
        ],
        "start_time": 0.8399999737739563,
        "type": "entity",
        "written_form": [{
                "alternatives": [{
                    "confidence": 1,
                    "content": "17th",
                    "language": "en"
                }],
                "end_time": 1.1999999682108562,
                "start_time": 0.8399999737739563,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "of",
                    "language": "en"
                }],
                "end_time": 1.559999962647756,
                "start_time": 1.1999999682108562,
                "type": "word"
            },
            {
```

```
                "alternatives": [{
                    "confidence": 1,
                    "content": "January",
                    "language": "en"
                }],
                "end_time": 1.9199999570846558,
                "start_time": 1.559999962647756,
                "type": "word"
            },
            {
                "alternatives": [{
                    "confidence": 1,
                    "content": "2022",
                    "language": "en"
                }],
                "end_time": 3.0899999141693115,
                "start_time": 1.9199999570846558,
                "type": "word"
            }
        ]
    }],
    "metadata": {
        "end_time": 5.16,
        "start_time": 0,
        "transcript": "17th of January 2022 "
    }
}]
```


If `enable_entities` is set to `false`, the output is as below:


```
  [{
    "message": "AddTranscript",
    "format": 2.7,
    "results": [{
            "alternatives": [{
                "confidence": 1,
                "content": "17th",
                "language": "en"
            }],
            "end_time": 1.1999999682108562,
            "start_time": 0.8399999737739563,
            "type": "word"
        },
        {
            "alternatives": [{
                "confidence": 1,
                "content": "of",
                "language": "en"
            }],
            "end_time": 1.559999962647756,
            "start_time": 1.1999999682108562,
            "type": "word"
        },
        {
```

```
            "alternatives": [{
                "confidence": 1,
                "content": "January",
                "language": "en"
            }],
            "end_time": 1.9199999570846558,
            "start_time": 1.559999962647756,
            "type": "word"
        },
        {
            "alternatives": [{
                "confidence": 1,
                "content": "2022",
                "language": "en"
            }],
            "end_time": 3.0899999141693115,
            "start_time": 1.9199999570846558,
            "type": "word"
        }
    ],
    "metadata": {
        "end_time": 5.16,
        "start_time": 0,
        "transcript": "17th of January 2022 "
    }
}]
```