SPEECHMATICS

Real-time Container 1.3.0

Table of Contents

- Real-time Container
  - High Level Summary
  - Important Notice
  - What's New
    - 1.3.0
  - Known Limitations
  - Resolved Issues
  - Supported Languages
  - Supported Platforms
  - Installation
  - Prerequisites
- Real-time Container Quick Start Guide
  - System Requirements
  - Architecture
  - Supported File Formats
  - Accessing the Image
    - Getting the Image
      - Software Repository Login
      - Pulling the Image
  - Licensing
  - Using the Container
    - Input Modes
      - Output
      - Transcription duration information
    - Running a Container in Read-Only Mode
    - Running a Container as a non-root user
    - How to use a Shared Custom Dictionary Cache
- Health service
  - Endpoints
    - `/started`
    - `/live`
    - `/ready`
- Troubleshooting
  - Enabling Logging
  - Licensing
  - Common Problems
- Real-time Container API Guide
  - Client ↔ API endpoint
    - Messages
      - StartRecognition
      - AddAudio
      - AudioAdded
        - Implementation details
      - AddTranscript
      - AddPartialTranscript
      - SetRecognitionConfig
      - EndOfStream
      - EndOfTranscript

            - Supported audio types
            - Transcription config
            - Additional words
            - Output locale
            - Punctuation overrides
            - Error messages
                - Error types
            - Warning messages
                - Warning types
            - Info messages
                - Info message types
        - Example communication
- Examples how to use the V2 API
    - WebSocket URI
    - Session Configuration
        - TranscriptionConfig
        - AddAudio
        - Final and Partial Transcripts
    - Advanced Punctuation
- Example Usage
    - JavaScript
    - Python
        - Standalone Real-Time Container Usage
- Formatting Common Entities
    - Overview
    - Supported Languages
    - Using the enable_entities parameter
    - Configuration example
    - Different entity classes
    - Output locale styling
    - Example output

Real-time Container

High Level Summary

This release updates the language packs for Swedish and Arabic. It also adds metadata tags for disfluencies in English in the JSON output and updates Linux Ubuntu OS on which the Container runs.

Important Notice

This release makes changes to the buffer when sending `AddAudio` messages. If an `AudioAdded` message is now sent, it means that the audio is definitively ready for transcription. The Real-time Container has a buffer of up to 10 seconds of speech, or 500 add Audio messages. If this buffer is exceeded, no further `AudioAdded` messages messages will be returned from the Container until the buffer has capacity. Please ensure any integration you have to the Real-time Container is able to tolerate this buffer by ensuring that sending and receiving messages runs in another thread or uses some other mechanism to avoid getting blocked. For the Real-time virtual Container where audio is sent faster than real-time, it is recommended to use a semaphore of size 500 or audio length of 10 seconds to avoid any unnecessary memory consumption.

It is recommended to run Speechmatics containers on processors that support Advanced Vector Extensions 2 (AVX2) in order to take advantage of latest performance optimisations.

What's New

1.3.0

- Improved Swedish and Arabic language packs, both now have advanced punctuation enabled (Swedish supports . ? , ! and Arabic supports . ؟ ، !)
- Disfluency tagging in English
- The API version has been updated to 2.6
- The Container now runs on Ubuntu Focal (OS 20.04)
- Speechmatics containers are now built using [Docker Buildkit](). If you have an internal registry to host the Speechmatics container which uses both Nexus and self-signed certificates, please make sure you are on [Nexus version 3.15 or above]() or you may encounter errors.

Known Limitations

The following are known issues in this release:

Issue ID	Summary	Detailed Description and Possible Workarounds
REQ-10634	Putting "-" as an item in `additional vocab` configuration will cause the container to fail	Do not enter just a "-" on its own in Custom Dictionary either as an additional vocab item or in the `sounds_like property`. Hyphens are still supported when entered as part of phrases or words
REQ-13240	Chinese (cmn) container crashes occasionally when using certain additional vocabulary	Do not use whitespace characters in additional vocabulary `sounds_like`
REQ-	Audio Swapping	Repeatedly audio swapping between 8kHz and 16kHz files can cause

16256	between 8kHz and 16kHz causes memory leak	an increase in memory over very long periods that causes the container to crash. If memory usage in this scenario becomes excessive it is recommended to restart the container
REQ-17771	Wide-space Unicode characters in Custom Dictionary cause a jobs to fail	This is now fixed and wide-spaced characters should be accepted

Resolved Issues

The following is a list of any resolved issues within this release

The following issues are addressed since the previous release:

Issue ID	Summary	Resolution Description
REQ-11135	Unwanted hesitations in transcripts.	For the English language pack Speechmatics now tags hesitation words ('umm') with a metadata tag of "disfluency". Users can use this tag for post-processing to filter or analyze such words. This work does not make disfluencies better or more poorly recognised in transcript output
REQ-11136	Transcripts are direct written to the Real-time Virtual Appliance and Container logs	Transcripts are no longer written directly to the logs or persisted to disk, even temporarily, for security reasons.
REQ-14795	Configuration information was not written to logs in StartRecognitionMessage	Transcription Configuration information is now logged as part of th StartRecognitionMessage. Individual custom dictionary entries are redacted
REQ-15515	Internal buffer limit of 500 AddAudio messages/10 seconds of audio	The Container now has a buffer. If you are sending audio faster than real-time and send more than 500 AddAudio messages of 10 seconds of Audio you will not receive an audioAdded response until there is capacity again. Please ensure your client connection is resilient to avoid audio being dropped

Supported Languages

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

For example,

```
rt-asr-transcriber-en:1.3.0
```

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 }} vCPU
- {{ 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). To take advantage of recent performance improvements, Speechmatics' deployments require processors that support at least Advanced Vector Extensions 2 (AVX2), and require as a minimum processors that support Advanced Vector Extensions (AVX) for the container to successfully run. 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 protocol, and will provide the following output:
    - Words in the transcript
    - Word confidence
    - Timing information
    - Relevant metadata 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.

Supported File Formats

Only the following file formats are supported:

- aac
- amr
- avi
- caf
- flac
- flv
- m4a
- m4v
- mkv
- mov
- mp3
- mp4
- mpg
- mpeg
- ogg
- wav
- wma
- wmv

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 credentials – provided by the Speechmatics team
- Speechmatics Docker URL - https://speechmatics-docker-public.jfrog.io
- 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, using a method such as the CLI

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-public.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-public.jfrog.io/rt-asr-transcriber-en:1.3.0
```

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

```
docker pull speechmatics-docker-public.jfrog.io/rt-asr-transcriber-es:1.3.0
```

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

:::important Docker Image Caching 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.*

As of Feb 2021, all Speechmatics containers are built using Docker Buildkit. This should not impact your internal management of the Speechmatics Container. If you use JFrog to host the Speechmatics container there may be

some UI issues see here, but these are cosmetic and should not impact your ability to pull and run the container. If your internal registry uses Nexus and self-signed certificates, please make sure you are on Nexus version 3.15 or above or you may encounter errors.

Licensing

You should have received a confidential license file from Speechmatics containing a token to use to license your container. The contents of the file received should look similar to this:

```
{
    "contractid": 1,
    "creationdate": "2020-03-24 17:43:35",
    "customer": "Speechmatics",
    "id": "c18a4eb990b143agadeb384cbj7b04c3",
    "is_trial": true,
    "metadata": {
        "key_pair_id": 1,
        "request": {
            "customer": "Speechmatics",
            "features": [
                "MAPRT",
                "LANY"
            ],
            "isTrial": true,
            "notValidAfter": "2021-01-01",
            "validFrom": "2020-01-01"
        }
    },
    "signedclaimstoken": "example",
}
```

The `validFrom` and `notValidAfter` keys in the license file specify the start and end dates for the validity of your license. The license is valid from 00:00 UTC on the start date to 00:00 UTC on the expiry date. After the expiry date, the container will continue to run but will not transcribe audio. You should apply for a new license before this happens.

Licensing does not require an internet connection.

There are two ways to apply the license to the container.

- As a volume-mapped file

The license file should be mapped to the path `/license.json` within the container. For example:

```
docker run --volume $PWD/my_license.json:/license.json:ro rt-asr-transcriber-en:1.3.0
```

- As an environment variable

Setting an environment variable named `LICENSE_TOKEN` is also a valid way to license the container. The contents of this variable should be set to the value of the `signedclaimstoken` from within the license file.

For example, copy the `signedclaimstoken` from the license file (without the quotation marks) and set the environment variable as below:

```
docker run -e LICENSE_TOKEN='example' rt-asr-transcriber-en:1.3.0
```

If both a volume-mapped file and an environment variable are provided simultaneously then the volume-mapped file will be ignored.

Using the Container

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

Here's an example of how to start the container from the command-line:

```
docker run -p 9000:9000 -p 8001:8001 -e LICENSE_TOKEN='example' rt-asr-transcriber-en:1.3.0
```

The Docker `run` options used are:

Name	Description
`--port, -p`	Expose ports on the container so that they are accessible from the host
`--env, -e`	Set the value of an environment variable

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": "2.6",
    "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 }
            ]
        }
    ]
}
```


Transcription duration information

The container will output a log message after every transcription session to indicate the duration of speech transcribed during that session. This duration only includes speech, and not any silence or background noise which was present in the audio. It may be useful to parse these log messages if you are asked to report usage back to us, or simply for your own records.

The format of the log messages produced should match the following example:

```
2020-04-13 22:48:05.312 INFO sentryserver Transcribed 52 seconds of speech
```

Consider using the following regular expression to extract just the seconds part from the line if you are parsing it:

```
^.+ .+ INFO sentryserver Transcribed (\d+) seconds of speech$
```

Running a Container in Read-Only Mode

Users may wish to run the container in read-only mode. This may be necessary due to their regulatory environment, or a requirement not to write any media file to disk. An example of how to do this is below.

```
docker run -it --read-only \
-p 9000:9000 \
--tmpfs /tmp \
-e LICENSE_TOKEN=$TOKEN_VALUE \
rt-asr-transcriber-en:1.3.0
```

The container still requires a temporary directory with write permissions. Users can provide a directory (e.g `/tmp`) by using the `--tmpfs` Docker argument. A tmpfs mount is temporary, and only persisted in the host memory. When the container stops, the tmpfs mount is removed, and files written there won't be persisted.

If customers want to use the shared custom dictionary cache feature, they must also specify the location of cache and mount it as a volume

```
docker run -it --read-only \
-p 9000:9000 \
--tmpfs /tmp \
-v /cachelocation:/cache \
-e LICENSE_TOKEN=$TOKEN_VALUE \
```

```
-e SM_CUSTOM_DICTIONARY_CACHE_TYPE=shared \
rt-asr-transcriber-en:1.3.0
```

Running a Container as a non-root user

A Real-time container can be run as a non-root user with no impact to feature functionality. This may be required if a hosting environment or a company's internal regulations specify that a container must be run as a named user.

Users may specify the non-root command by the `docker run --user $USERNUMBER:$GROUPID`. User number and group ID are non-zero numerical values from a value of **1** up to a value of **65535**

An example is below:

```
docker run -it --user 100:100 \
-p 9000:9000 \
-e LICENSE_TOKEN=$TOKEN_VALUE \
rt-asr-transcriber-en:.1.3.0
```

How to use a Shared Custom Dictionary Cache

For more information on how the Custom Dictionary works, please see the Speech API Guide.

The Speechmatics Real-time Container includes a cache mechanism for custom dictionaries to improve set-up performance for repeated use. By using this cache mechanism, transcription will start more quickly when repeatedly using the same custom dictionaries. You will see performance benefits on re-using the same custom dictionary from the second time onwards.

It is not a requirement to use the shared cache to use the Custom Dictionary.

The cache volume is safe to use from multiple containers concurrently if the operating system and its filesystem support file locking operations. The cache can store multiple custom dictionaries in any language used for transcription. It can support multiple custom dictionaries in the same language.

If a custom dictionary is small enough to be stored within the cache volume, this will take place automatically if the shared cache is specified.

For more information about how the shared cache storage management works, please see **Maintaining the Shared Cache**.

We highly recommend you ensure any location you use for the shared cache has enough space for the number of custom dictionaries you plan to allocate there. How to allocate custom dictionaries to the shared cache is documented below.

How to set up the Shared Cache

The shared cache is enabled by setting the following value when running transcription:

- Cache Location: You must volume map the directory location you plan to use as the shared cache to `/cache` when submitting a job
- `SM_CUSTOM_DICTIONARY_CACHE_TYPE` : (mandatory if using the shared cache) This environment variable must be set to `shared`
- `SM_CUSTOM_DICTIONARY_CACHE_ENTRY_MAX_SIZE` : (optional if using the shared cache). This determines the maximum size of any single custom dictionary that can be stored within the shared cache in **bytes**
  - E.G. a `SM_CUSTOM_DICTIONARY_CACHE_ENTRY_MAX_SIZE` with a value of 10000000 would set a max storage size of any custom dictionary at **10MB**
  - For reference a custom dictionary wordlist with 1000 words produces a cache entry of size around 200 kB, or **200000** bytes

- A value of `-1` will allow **every** custom dictionary to be stored within the shared cache. This is the **default** assumed value
- A custom dictionary cache entry **larger** than the `SM_CUSTOM_DICTIONARY_CACHE_ENTRY_MAX_SIZE` will still be used in transcription, but will not be cached

Maintaining the Shared Cache

If you specify the shared cache to be used and your custom dictionary is within the permitted size, Speechmatics Real-time Container will always try to cache the custom dictionary. If a custom dictionary cannot occupy the shared cache due to other cached custom dictionaries within the allocated cache, then older custom dictionaries will be removed from the cache to free up as much space as necessary for the new custom dictionary. This is carried out in order of the least recent custom dictionary to be used.

Therefore, you must ensure your cache allocation large enough to handle the number of custom dictionaries you plan to store. We recommend a relatively large cache to avoid this situation if you are processing multiple custom dictionaries using the batch container (e.g 50 MB). If you don't allocate sufficient storage this could mean one or multiple custom dictionaries are deleted when you are trying to store a new custom dictionary.

It is recommended to use a docker volume with a dedicated filesystem with a limited size. If a user decides to use a volume that shares filesystem with the host, it is the user's responsibility to purge the cache if necessary.

Creating the Shared Cache

In the example below, transcription is run where an example local docker volume is created for the shared cache. It will allow a custom dictionary of up to 5MB to be cached.

```
docker volume create speechmatics-cache

docker run --rm -d \
  -p 9000:9000 \
  -e LICENSE_TOKEN='example' \
  -e SM_CUSTOM_DICTIONARY_CACHE_TYPE=shared \
  -e SM_CUSTOM_DICTIONARY_CACHE_ENTRY_MAX_SIZE=5000000 \
  -v speechmatics-cache:/cache \
  rt-asr-transcriber-en:1.3.0

speechmatics transcribe --additional-vocab gnocchi --url ws://localhost:9000/v2 --ssl-mode=none test.mp3
```

Viewing the Shared Cache

If all set correctly and the cache was used for the first time, a single entry in the cache should be present.

The following example shows how to check what Custom Dictionaries are stored within the cache. This will show the **language**, the **sampling rate**, and the **checksum** value of the cached dictionary entries.

```
ls $(docker inspect -f "{{.Mountpoint}}" speechmatics-cache)/custom_dictionary
en,16kHz,bef53e5bcca838a39c3707f1134bda6a09ff87aaa092036175287747344550edd
```

Reducing the Shared Cache Size

Cache size can be reduced by removing some or all cache entries.

```
rm -rf $(docker inspect -f "{{.Mountpoint}}" speechmatics-cache)/custom_dictionary/*
```

:::note Manually purging the cache Before manually purging the cache, ensure that no containers have the volume mounted, otherwise an error during transcription might occur. Consider creating a new docker volume as a

temporary cache while performing purging maintenance on the cache. :::

Health service

The container is able to expose an HTTP health service, which offers startup, liveness and readiness probes. This is accessible from Port 8001, and has 3 endpoints, `started`, `live` and `ready`. These can be used to see whether all services in the container are running or active respectively. 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/).

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:

`/started`

This endpoint provides a startup probe. It can be queried using an HTTP GET request. You must include the relevant port, 8001, in the 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:

```
$ curl -i address.of.container:8001/started
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.8.5
Date: Mon, 08 Feb 2021 12:46:21 GMT
Content-Type: application/json
{
    "started": true
}
```

`/live`

This endpoint provides a liveness probe. It can be queried using an HTTP GET request. You must include the relevant port, 8001, in the 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:

```
$ curl -i address.of.container:8001/live
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.8.5
Date: Mon, 08 Feb 2021 12:46:45 GMT
Content-Type: application/json
{
    "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:

```
$ curl -i address.of.container:8001/ready
HTTP/1.0 200 OK
Server: BaseHTTP/0.6 Python/3.8.5
Date: Mon, 08 Feb 2021 12:47:05 GMT
Content-Type: application/json
{
    "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 rt-asr-transcriber-en:1.3.0
```

Licensing

The best way to identify licensing errors with the container is to look at the container logs. See https://docs.docker.com/config/containers/logging/ for more information about doing this. If licensing is successful then the logs upon startup should look similar to this:

```
INFO:__main__:Starting health service
INFO:orchestrator.health:Health check server starting...
INFO:__main__:Health service started.
INFO:orchestrator.license:Starting sentry server...
time="2020-03-27T11:50:18.9774596Z" level=info msg="Listening to port 52000, secure mode = false"
time="2020-03-27T11:50:18.9776369Z" level=info msg="Reading license from /license.json"
time="2020-03-27T11:50:18.9866595Z" level=info msg="Read token eyJkbGciOjJS..."
INFO:orchestrator.license:Sentry server started
time="2020-03-27T11:50:18.990334Z" level=info msg="License : licensed=true, customer=Speechmatics, contract_id=0, expires_at=2021-03-16 00:00:00 +0000 UTC, trial=false, features=MAPRT,MAPBA,AMCC,APD,APR,ASS"
time="2020-03-27T11:50:18.9904803Z" level=info msg="Starting server 3.0.0 [master]"
time="2020-03-27T11:50:18.9918058Z" level=info msg="Monitoring parent pid 1"
2020-03-27 11:50:19,005 orchestrator.transport.ws.common        INFO    Waiting for the model to be ready - checking /model/manifest.json
2020-03-27 11:50:20,673 orchestrator.transport.ws.common        INFO    Loading model en
2020-03-27 11:50:26,107 orchestrator.transport.ws.ws    INFO    transport websocket listening at ws://0.0.0.0:9000
2020-03-27 11:50:26,107 orchestrator.transport.ws.health_update INFO    Transport marked as started for health updates.
```

If your container is not licensed, or has an invalid license then it will exit upon startup with an error message similar to this:

```
RuntimeError: Failed to launch sentry server licensing process on port 52000
```

Please ensure that you have correctly followed the instructions in the quick start guide for setting up licensing, and that have you a license file which has not expired (the `metadata` section in the file tells you when the license is valid until).

There can be several reasons for a licensing error:

- No license has been provided

If you see the following message in the container logs then the most likely cause is that no license file has been provided:

```
level=error msg="could not load license file data: stat /license.json: no such file or directory"
```

Please review the quick start guide and ensure that the license has been provided properly, either as a volume-mapped file or as an environment variable.

- The license has expired

```
level=info msg="License : licensed=false, customer=Speechmatics, contract_id=99,
expires_at=2020-03-26 00:00:00 +0000 UTC, trial=false, features="
level=error msg="Error in license : token is expired by 36h6m37s"
```

This message indicates that your license has expired. Please request a new license from Speechmatics support.

- You are attempting to use a feature for which you are not licensed

Not all licenses are valid for all features of our product. If you are not licensed for a feature which you attempt to use for transcription, then transcription will not be performed. Please get in touch with Speechmatics support if you are interested in using a feature which you are not licensed for.

If this error case happens you should see a log message similar to this one:

```
2020-03-27 12:11:04,230 orchestrator.transport.ws.protocol      WARNING Sending an error to
client: not_allowed - Unable to use provided configuration: No license for requested
language - LEN; session ID de1ec62d-a22d-47a3-8f03-def025a52f60
```

- An improperly formatted license file has been provided

Only relevant if using a volume-mapped file to license the container

```
level=error msg="could not load license file data: unexpected end of JSON input"
```

or

```
level=error msg="could not load license file data: No valid signedclaimstoken field found
in license (too short)"
```

Please ensure that you are using the license file which has been provided to you by the Speechmatics support team, and that no changes have been made to the file accidentally.

The license file should be a valid JSON file and should contain a key named `signedclaimstoken` which is your license token.

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 2.6

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",
        "additional_vocab": ["gnocchi", "bucatini", "bigoli"],
        "diarization": "speaker_change",
        "enable_partials": true,
        "punctuation_overrides": {
            "permitted_marks": [",", "."]
        }
```

```
    }
}
```

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', '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. '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.
    - `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.

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.

SetRecognitionConfig

Allows the **Client** to configure the recognition session even after the initial `StartRecognition` message. **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`
- `enable_partials`

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

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"`.

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).

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.

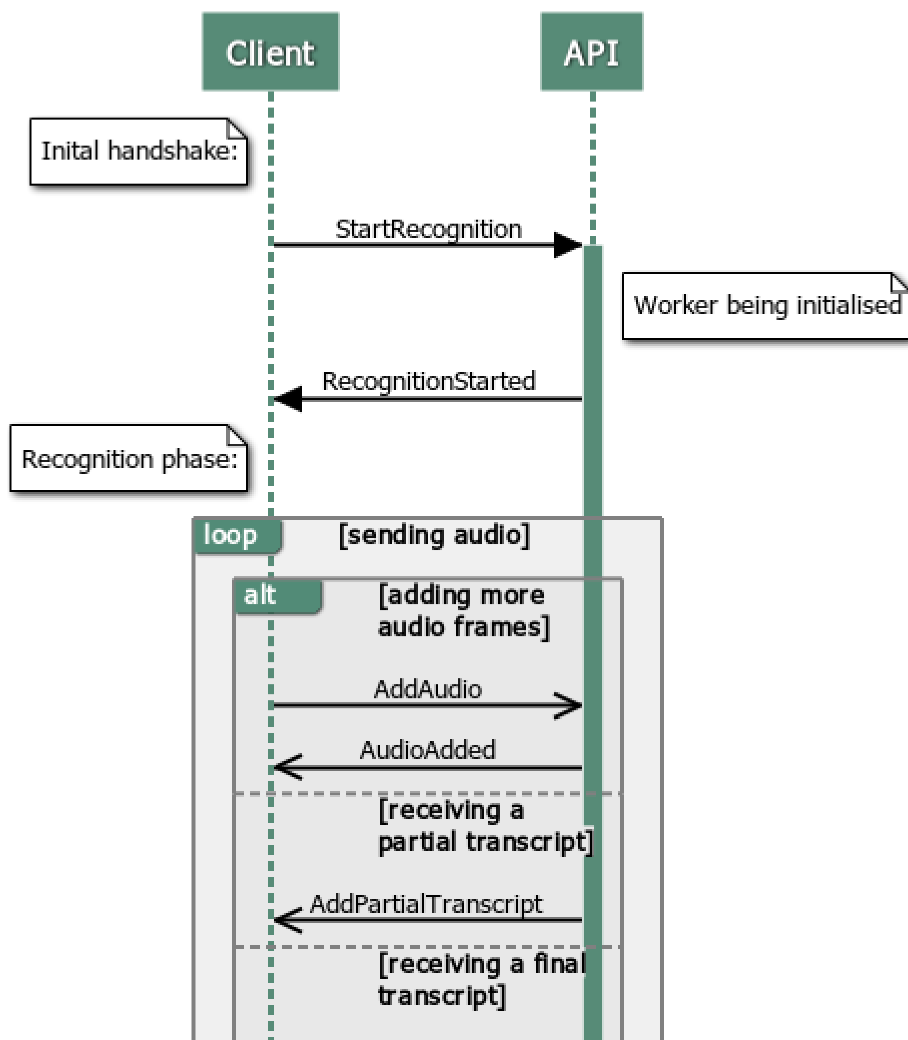
Afterwards, the **transcription** stage happens. The client keeps sending `AddAudio` messages. The API asynchronously replies with `AudioAdded` messages. The API also asynchronously sends `AddPartialTranscript` and `AddTranscript` messages.

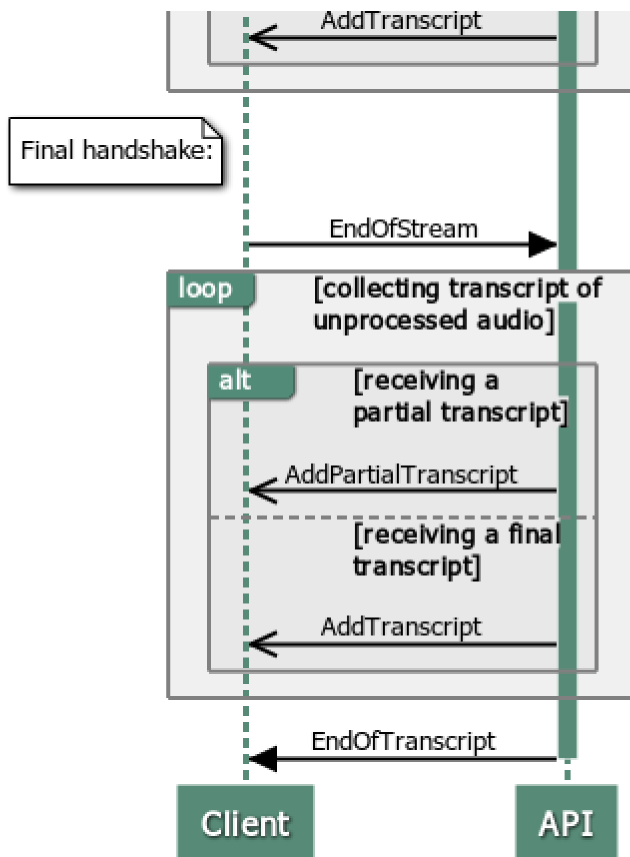
Once the client doesn't want to send any more audio, the **disconnect handshake** is performed. The Client sends an `EndOfStream` message as it's last message. No more messages are handled by the API afterwards. The API processes whatever audio it has buffered at that point and sends all the `AddTranscript`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

:::info WebSocket Schemes 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":"2.6"
}
```

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 2.6. This may change in future releases if the output format is updated.

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`. Please contact support@speechmatics.com if you require this library. 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 alphanumerics, where an alphanumeric is a digit less than 10, a character or symbol	triple seven five four	77754
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	one hundred to two	100 to £200 million

	and y correspond to another entity class	hundred million pounds	
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 "
    }
}]
```