SPEECHMATICS

Batch Container 6.3.0

## Table of Contents

# Batch Container

## High-Level Summary

This release contains instructions on how to run the container with a parallelisation parameter to boost performane on multi-CPU platforms. It also contains improvements to telephony models for English and Spanish, and a number of bugfixes.

The following languages now support advanced punctuation: English (en), German (de), Spanish (es), French (fr), Dutch (nl), Danish (da), Turkish (tr), Malay (ms).

It is recommended that customers on previous releases upgrade to this version.

### Important Notices

The legacy V1 API and related output formats will be discontinued in a future release as we align the product with the same V2 API used by the Speechmatics SaaS: https://asr.api.speechmatics.com/v2/docs. We recommend that customers familiarise themselves with the configuration object used to specify job configurations as any new features will only be supported using this mechanism. Future notices will be provided to announce the end of life of the V1 API, and provide detailed instructions on migrations to the V2 API.

### What's New

- SubRip (srt) subtitle format
- The ability to transcribe in parallel across multiple CPUs
- Improvements to the telephony models for English and Spanish

### Issues Fixed

The following issues are addressed since the previous release:

| Issue ID | Summary | Resolution Description |
|---|---|---|
| REQ-10377, REQ-9569 | Unfriendly error when empty audio file is submitted | It is now possible to submit zero length audio files. |
| REQ-11728 | Files less than 1 second skip decoding. | Files that are < 0.3s long are considered to be zero length; but files between 0.3 and 1.0s (and above) will now be transcribed (assuming speech is detected in them). |
| REQ-10095 | StopIteration Exception in post processing can cause a job to fail | The exception is now handled properly. |

### Known Limitations

| Issue ID | Summary | Detailed Description and Possible Workarounds |
|---|---|---|
| REQ-1409 | Proteus HCL with `<unk>` causes out of memory error | A custom dictionary list that contains the word '' causes the worker to crash. |
| | | |

| | | |
|---|---|---|
| REQ-10160 | Advanced punctuation for Spanish (es) does not contain inverted marks. | Inverted marks [ ¿ ¡ ] are not currently available for Spanish advanced punctuation. |
| REQ-10627 | Double full stops when acronym is at the end of the sentence | If there is an acronym at the end of the sentence, then a double full stop will be output, for example: "team G.B.." |
| REQ-11135 | A previous release (6.1.0) introduced unwanted hesitations in transcripts. | Due to changes in the way that training data is now ingested to improve the accuracy of spontaneous speech for English (en) there is a greater likelihood that hesitations will be included in the output transcrtips. We plan to support a hesitation filtering capability in a future release for customers that do not want to see hesitations on transcripts. |

### Supported Platforms

Docker (17.06.0+) running on Ubuntu, Debian, Fedora or CentOS.

### Upgrade Path

None.

### Installation

Pull the Batch Container Docker image from the Speechmatics Docker repository.

### Pre-requisites

You have a login (URL, username and password) for the Speechmatics Docker repository, and have a Docker environment (version 17.06.0 or above) running.

### Related Documentation

- Speechmatics Batch Container Quick Start Guide version 6.2.0
- Speechmatics Batch Container API Guide version 6.2.0

For a complete list of languages that are supported by the Speechmatics Container, including those which have custom dictionary support, please go to the Speechmatics website: [https://www.speechmatics.com/language-support/](https://www.speechmatics.com/language-support/)

## Batch Container Quick Start Guide

This guide will walk you through the steps needed to deploy the Speechmatics Batch 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 Speechmatics container transcription service is detailed in the Speechmatics Container API guide.

## System requirements

Speechmatics containerized deployments are built on the Docker platform. In order to operate the containers, the following requirements will need to be met.

### Host requirements

An individual Docker image is required for each language transcription is required within. A single image can be used to create and run multiple containers concurrently, each running container will require the following resources:

- 1 vCPU
- 2-5GB RAM
- 100MB 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.

**Note:** Each language pack required is distributed as a separate Docker image. Only the language packs required need to be installed on the Docker host.

## Architecture

Each container:

- Provides the ability to transcribe recorded speech in a predefined language. The container will receive input from most audio and video formats, and will provide the following output:
  - Transcript word
  - Word confidence
  - Timing information
  - Speaker change and labelling information
- Takes one input file and outputs the resulting transcript
- Can run in a mode that parallelises processing across multiple processor cores
- Supports input file sizes up to 2 hours in length or 4GB in size
- All data is transitory, once a container completes its transcription it removes all record of the operation, no data is persisted.

In addition, 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.

## Accessing the Image

The Speechmatics Docker image is obtained 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
- Language Code – the ISO language code (for example `fr` for French)
- `LICENSE_KEY` - which is required to start a container
- `TAG` – which is used to identify 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 language pack required.

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

```
docker pull speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0
```

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

```
docker pull speechmatics-docker-example.jfrog.io/transcriber-es:6.3.0
```

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

The Docker images we provide have a configured expiry date and must be used in conjunction with the license key that has been issued to you. The Docker images and license key are specific to your organisation, and should not be shared with any third parties. License keys must be provided at runtime through a `LICENSE_KEY` environment value, like this:

```
-e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702
```

## 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.

There are two different methods for passing an audio file into a container:

- STDIN: Streams audio file into the container though the standard command line entry point
- File Location: Pulls audio file from a file location

Here are some examples below to demonstrate these modes of operating the containers.

Example 1: passing a file using the `cat` command to the Spanish (es) container

```
cat ~/sm_audio.wav | docker run -i \
  -e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-es:6.3.0
```

Example 2: pulling an audio file from a volume-ma directory into the container

```
docker run -i -v ~/sm_audio.wav:/input.audio \
  -e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-es:6.3.0
```

**NOTE:** the audio file must be mapped into the container with `:/input.audio`

The Docker `run` options used are:

| Name | Description |
|---|---|
| `--env, -e` | Set environment variables |
| `--interactive , -i` | Keep STDIN open even if not attached |
| `--volume , -v` | Bind mount a volume |

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

Both the methods will produce the same transcribed outcome. STDOUT is used to provide the transcription in a JSON format. Here's an example:


```
{
  "format": "1.0",
  "license": "docker-example build (Mon Nov 12 12:00:00 2019): 122 days remaining",
  "speakers": [
    {
      "duration": "2.28",
      "name": "UU",
      "time": "0.57"
    }
  ],
  "words": [
    {
      "confidence": "1.00",
      "duration": "0.51",
      "name": "This",
      "time": "0.57"
    },
    {
      "confidence": "1.00",
      "duration": "0.21",
      "name": "is",
```

```
      "time": "1.17"
    },
    {
      "confidence": "1.00",
      "duration": "0.09",
      "name": "a",
      "time": "1.38"
    },
    {
      "confidence": "1.00",
      "duration": "0.54",
      "name": "quick",
      "time": "1.47"
    },
    {
      "confidence": "1.00",
      "duration": "0.60",
      "name": "test",
      "time": "2.22"
    },
    {
      "confidence": "1.00",
      "duration": "0.03",
      "name": ".",
      "time": "2.82"
    }
  ]
}
```


### Determining success

The exit code of the container will determine if the transcription was successful. There are two exit code possibilities:

- Exit Code == 0 : The transcript was a success; the output will contain a JSON output defining the transcript (more info below)
- Exit Code != 0 : the output will contain a stack trace and other useful information. This output should be used in any communication with Speechmatics support to aid understanding and resolution of any problems that may occur

### Troubleshooting

#### Enabling Logging

If you are seeing problems then we recommend that you enable logging and open a support ticket with Speechmatics support: support@speechmatics.com.

To enable logging you add two environment variables:

- `SM_JOB_ID` - a job id, for example: `1`
- `SM_LOG_DIR` - the directory inside the container where to write the logs, for example: `/logs`

The following example shows how to do this, using the `-stderr=true` argument to dump the logs to stderr:

```
docker run --rm -e SM_JOB_ID=123 -e SM_LOG_DIR=/logs \
  -v ~/sm_audio.wav:/input.audio \
  -e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 \
```

```
  speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0 /
  -stderr=true
```

When raising a support ticket it is normally easier to write the log output to a specific file. You can do this by creating a volume mount where the logs will be accessible from after the container has finished. Before running the container you need to create a directory for the log file and ensure it has the correct permissions. In this example we use a local logs directory to store the output of the log for a job with ID 124:

```
mkdir -p logs/124
sudo chown -R nobody:nogroup logs/
sudo chmod -R a+rwx logs/

docker run --rm -v ${PWD}/logs:/logs -e SM_JOB_ID=124 -e SM_JOB_ID=/logs \
  -v ~/sm_audio.wav:/input.audio \
  -e LICENSE_KEY=f787b0051e2768b1f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0

tail logs/124/sigurd.log
```

#### Common Problems

Common problems to look out for are when there is no recognisable audio in the input file, or it is in a format that is unsupported. You should also ensure, when using the config object, that the JSON is correctly formatted.

## Building an Image

Using STDIN to pass files in and obtain the transcription may not be sufficient for all use cases. It is possible to build a new Docker Image that will use the Speechmatics Image as a layer. This will allow greater flexibility and a mechanism to fit into custom workflows. To include the Speechmatics Docker Image inside another image, ensure to add the pulled Docker image into the Dockerfile for the new application.

### Requirements for a custom image

To ensure the Speechmatics Docker image works as expected inside the custom image, please consider the following:

- Any audio that needs to be transcribed must to be copied to a file called "/input.audio" inside the running container
- To initiate transcription, call the application `pipeline`. The pipeline will start the transcription service and use `/input.audio` as the audio source
- Once pipeline finishes transcribing, ensure you move the transcription data outside the container
- Shutdown the container after each transcription of an audio file

#### Dockerfile

To add a Speechmatics Docker image into a custom one, the Dockerfile must be modified to include the full image name of the locally available image.

Example: Adding Global English (en) with tag {{ book.product.version }} to the Dockerfile

```
FROM speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0
ADD download_audio.sh /usr/local/bin/download_audio.sh
RUN chmod +x /usr/local/bin/download_audio.sh
CMD ["/usr/local/bin/download_audio.sh"]
```

Once the above image is built, and a container instantiated from it, a script called `download_audio.sh` will be executed (this could do something like pulling a file from a webserver and copying it to `/input.audio` before

starting the pipeline application). This is a very basic Dockerfile to demonstrate a way of orchestrating the Speechmatics Docker Image.

**NOTE:** For support purposes, it is assumed the Docker Image provided by Speechmatics has been unmodified. If you experience issues, Speechmatics support will require you to replicate the issues with the unmodified Docker image e.g. `speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0`

# Batch Container API Guide

This guide will walk you through using the API's used to invoke features of the Speechmatics Batch Container.

For information on getting started and accessing the Speechmatics software repository please refer to Speechmatics Container Quick Start Guide.

## Transcription Output Format

The transcript output will consist of:

- Diarization information
  - Channel Diarization - channel labelling with relevant transcription in enclosed block
  - Speaker Diarization - speaker identification, speakers (F# and M#) are identified at the beginning of the transcript with timing information
  - No diarization - unidentified speakers will be displayed as "UU" at the beginning of the transcript
- JSON format version (examples can be seen in the sections below)
  - 1.0 - used for transcriptions with Speaker Diarization
  - 1.1 - used for transcriptions when Channel Diarization is enabled
  - 2.4 - used when the config.json configuration object is used (recommended approach)
- Header information to show license expiry date
- A full stop to delimit sentences, irrespective of language being transcribed
- A word, confidence and timing information for each transcribed word

## Feature Usage

This section explains how to use additional features beyond plain transcription of speech to text.

There are two methods that can be used to access these additional features:

- Use of environment values and word lists
- Passing a configuration file that contains all feature settings

Of these, the second method is preferable as it allows all feature settings to be supplied as part of one object, and is more extensible for future use. Both methods are descibed in this guide.

### Configuration Object

The config object, if used, is a JSON structure that can be passed as a volume-mapped file (mapped to `/config.json`) like this:

```
docker run -i -v ~/Projects/ba-test/data/shipping-forecast.wav:/input.audio \
  -v ~/tmp/config.json:/config.json \
  speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0
```

A simple example of a config object file (`~/tmp/config.json` from the above example) providing custom dictionary words as part of the `additional_vocab` property is shown below:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "additional_vocab": ["Met Office", "Fitzroy", "Forties"]
  }
}
```

The transcript will be in the version 2.4 JSON format, for example:

```
{
  "format": "2.4",
  "metadata": {
    "created_at": "2019-03-01T17:21:34.002Z",
    "type": "transcription",
    "transcription_config": {
      "language": "en",
      "diarization": "none",
      "additional_vocab": [
        {
          "content": "Met Office"
        },
        {
          "content": "Fitzroy"
        },
        {
          "content": "Forties"
        }
      ]
    }
  },
```

## Diarization

Diarization is the ability to identify a speaker in an audio file. This identification is only related to a single audio file only. For audio files that contain multiple channels or streams, it is possible to use **channel diarization** and apply custom labels to each channel or stream. If your audio file contains only a single channel or stream then you should choose **speaker diarization**. By default, containers will transcribe a file with diarization *disabled*.

**Note:** Enabling this feature increases the amount of time taken to transcribe an audio file.

### Speaker Diarization

To enable speaker diarization the following must be set if you are using the config object:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "diarization": "speaker"
  }
}
```

Or, if you are using environment variables like this:

```
-e DIARIZE=true
```

When enabled, the output will contain the speaker Identifiers, these are explained below:

- `M#` - Identifies a male speaker. The # will be a number identifying an individual male speaker
- `F#` - identified a female speaker. The # will be a number identifying an individual female speaker
- `UU` - Speaker is not identified (or diarization is disabled)

Example enabling speaker diarization:

```
docker run -i -v /home/user/sm_audio.wav:/input.audio \
  -e DIARIZE=true \
  -e LICENSE_KEY=f787b0051e2768bcee3231f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0
```

Example response to show speaker labelling (transcript is not shown):


```
{
  "format": "1.0",
  "license": "docker-example build (Wed Oct 27 12:23:00 2018): 122 days remaining",
  "speakers": [
  {
      "duration": "10.43",
      "name": "F2",
      "time": "7.82"
    },
    {
      "duration": "20.84",
      "name": "M2",
      "time": "18.41"
    },
    {
      "duration": "193.02",
      "name": "M1",
      "time": "41.45"
    },
    {
      "duration": "263.54",
      "name": "M2",
      "time": "234.47"
    },
    {
      "duration": "81.96",
      "name": "M1",
      "time": "498.06"
    },
    {
      "duration": "22.14",
      "name": "M2",
      "time": "581.63"
    },
    {
      "duration": "193.93",
      "name": "M1",
      "time": "604.55"
    },
```

```
    {
      "duration": "1.69",
      "name": "F5",
      "time": "803.48"
    }
  ],
}
```

Timing information identifies when the speaker started to talk.

#### Channel Diarization

Channel diarization allows individual channels in an audio file to be labelled. This is ideal for audio files with multiple channels (up to 6). By default the feature is disabled. The following is required to enable channel diarization on a 2-channel file that will use labels `Customer` and `Agent`:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "diarization": "channel",
    "channel_diarization_labels": ["Customer", "Agent"]
  }
}
```

If the config object file is called `config.json` then you would start the transcription job like this:

```
docker run -i -v ~/Projects/ba-test/data/shipping-forecast.wav:/input.audio \
  -v ~/tmp/config.json:/config.json \
  speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0
```

Or, using environment variables like this:

```
-e CHANNEL_DIARIZATION=true
```

Channel labels are optional and if provided must be comma separated ASCII characters, for example:

```
-e CHANNEL_DIARIZATION_LABELS=Agent,Caller
```

Here is an example using channel diarization with labels, passing in the settings as environment variables:

```
docker run -i -v /home/user/sm_audio.wav:/input.audio \
  -e CHANNEL_DIARIZATION=true -e CHANNEL_DIARIZATION_LABELS=Agent,Caller \
  -e LICENSE_KEY=f787b0051e2768bcee3231f619d75faab97f23ee9b7931890c05f97e9f550702 \
  speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0
```

For each named channel, the words will be listed in its own labelled block, for example:

```
{
  "channels": {
    "Agent": {
      "words": [
        {
          "confidence": "1.00",
          "duration": "0.18",
          "name": "Hey",
```

```
          "time": "4.38"
        },
        {
          "confidence": "1.00",
          "duration": "0.10",
          "name": "how's",
          "time": "4.65"
        },
        {
          "confidence": "1.00",
          "duration": "0.07",
          "name": "it",
          "time": "4.78"
        },
        {
          "confidence": "1.00",
          "duration": "0.27",
          "name": "going",
          "time": "4.96"
        }
        {
          "confidence": "1.00",
          "duration": "1.43",
          "name": ".",
          "time": "5.73"
        }
      ]
    },
    "Caller": {
      "words": [
        {
          "confidence": "1.00",
          "duration": "0.17",
          "name": "Yeah",
          "time": "6.43"
        },
        {
          "confidence": "1.00",
          "duration": "0.11",
          "name": "going",
          "time": "6.63"
        },
        {
          "confidence": "1.00",
          "duration": "0.27",
          "name": "good",
          "time": "6.90"
        }
      ]
    }
  },
  "format": "1.1",
  "license": "docker-example build (Wed Nov 3 12:00:00 2018): 122 days remaining"
}
```

**Note:**

- `DIARIZE` and `CHANNEL_DIARIZATION` cannot both be set to `true` at the same time
- If `CHANNEL_DIARIZATION` is set to `true`, and no labels are defined with `CHANNEL_DIARIZATION_LABELS` then default labels will be used (channel_1, channel_2 etc)
- Spaces cannot be used in the channel labels

### Custom Dictionary

This allows a custom dictionary wordlist to be added to the container at runtime. Having additional words can improve the likelihood it will be output in the final transcription. For any audio file being transcribed one custom dictionary can be provided.

Prior to using this feature, consider the following requirements:

- File encoding must be UTF-8
- Maximum number of words or phrases in the list is 1000
- Each word or phrase needs to be on an individual line
- You should remove empty lines, and lines that contain only a hyphen ('-') character
- If want to add acronyms (for example 'CEO'), you should use the sounds feature (see below)

Example small custom dictionary:

```
speechmagic
supercalifragilisticexpialidocious
Techcrunch
Yahoo! Answers
```

To enable this feature, you use the `additional_vocab` property of the config object:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "additional_vocab": [
      "speechmagic",
      "supercalifragilisticexpialidocious",
      "Techcrunch",
      "Yahoo! Answers"
    ]
  }
}
```

The Custom Dictionary feature supports the Sounds extension that allows you to pass alternate pronunciations to words. For example the phrases "North Utsire" and "South Utsire" could be added as follows:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "additional_vocab": [
      { "content": "North Utsire", "sounds_like": ["North at Sierra"]},
      { "content": "South Utsire", "sounds_like": ["South at Sierra"]},
      "Fitzroy",
      "Forties",
      { "content": "CEO", "sounds_like": ["C.E.O."]}
    ]
```

```
  }
}
```

You can also pass the custom words list has to be passed as a volume-mapped file called extra_words.txt, like this:

```
-v /home/user/sm_dictionary01.txt:/extra_words.txt
```

If you do this, then the extra words file must be list of words or phrases, for example:

```
Fisher
Dogger
German Bight
```

**Note:** There is no ability to use Sounds when specifying an extra words list like this.

Here is an example custom dictionary passed to the transcription service:

```
docker run -i -v /home/user/sm_audio.wav:/input.audio \
  -v /home/user/sm_dictionary01.txt:/extra_words.txt \
  -e LICENSE_KEY=f787b0051e2768bcee3231f619d75faab97f23ee9b7931890c05f97e9f550702
  speechmatics-docker-example.jfrog.io/transcriber-en:6.3.0
```

Example response:


```
{
  "format": "1.0",
  "license": " docker-example build (Fri Nov 16 12:00:00 2019): 122 days remaining",
  "speakers": [
    {
      "duration": "4.83",
      "name": "UU",
      "time": "0.90"
    }
  ],
  "words": [
    {
      "confidence": "1.00",
      "duration": "1.23",
      "name": "Yahoo! Answers",
      "time": "0.90"
    },
    {
      "confidence": "1.00",
      "duration": "0.18",
      "name": "is",
      "time": "2.22"
    },
    {
      "confidence": "1.00",
      "duration": "0.09",
      "name": "a",
      "time": "2.40"
    },
    {
```

```
      "confidence": "1.00",
      "duration": "0.57",
      "name": "community",
      "time": "2.49"
    },
    {
      "confidence": "1.00",
      "duration": "0.54",
      "name": "driven",
      "time": "3.06"
    },
    {
      "confidence": "1.00",
      "duration": "0.57",
      "name": "question",
      "time": "3.60"
    },
    {
      "confidence": "1.00",
      "duration": "0.18",
      "name": "and",
      "time": "4.23"
    },
    {
      "confidence": "1.00",
      "duration": "0.54",
      "name": "answer",
      "time": "4.41"
    },
    {
      "confidence": "0.98",
      "duration": "0.71",
      "name": "website",
      "time": "5.02"
    },
    {
      "confidence": "1.00",
      "duration": "0.00",
      "name": ".",
      "time": "5.73"
    }
  ]
}
```


**Note:** From the above response, notice how `"Yahoo! Answers"` appears as one `"name"` value in the output. This is the expected behavior when a custom dictionary contains a multi-word phrase.

### Output Locale

It is possible to optionally specify the language locale to be used when generating the transcription output, so that words are spelled correctly, for cases where the model language is generic and doesn't already imply the locale. The `output_locale` configuration setting is used for this. As an example, the following configuration uses the Global English (en) language pack with an output locale of British English (en-GB):

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "output_locale": "en-GB"
  }
}
```

Currently, Global English is the only language pack that supports different output locales.

### Advanced Punctuation

Some language models now support advanced punctuation (see the Release Notes for a complete list). This uses machine learning techniques to add in more naturalistic punctuation to make the transcript more readable. As well as putting punctuation marks in more naturalistic positions in the output, additional punctuation marks such as commas (,) and exclamation and question marks (!, ?) will also appear.

There is no need to explicitly enable this in the job 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` settings in the config.json file, for example:

```
{
  "type": "transcription",
  "transcription_config": {
    "language": "en",
    "punctuation_overrides": {
      "permitted_marks": [".", ","]
    }
  }
}
```

Both plain text and JSON output supports punctuation. 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.

A sample JSON output containing punctuation looks like this:

```
{
  "alternatives": [
    {
      "confidence": 1,
      "content": ",",
      "language": "en",
      "speaker": "UU"
    }
  ],
  "attaches_to": "previous",
  "end_time": 10.15,
  "is_eos": false,
  "start_time": 10.15,
  "type": "punctuation"
}
```

**Note:** Advanced punctuation is a V2 feature so, only the V2 output format will show advanced punctuation marks.

If you specify the punctuation_overrides element for languages that do not yet support advanced punctuation then it is ignored.

### SubRip Subtitles

In addition to previously described JSON and text output formats, transcripts can be generated in the SubRip (srt) subtitle format. This can be done when using a mapped `config.json` and passing `--all-formats` flag during invocation. The transcripts will be saved in the working directory in all supported formats: JSON, text and SubRip.

The produced SubRip subtitles can be customised using configuration options described below:

```
{
  "type": "transcription",
  "transcription_config": {
    ...
  },
  "output_config": {
    "srt_overrides": {
      "max_line_length": 37,
      "max_lines": 2
    }
  }
}
```

- `max_line_length` : sets maximum count of characters per subtitle line including white space (default: `37` ).
- `max_lines` : sets maximum count of lines in a subtitle section (default: `2` ).

../common/parallel-processing.md

# 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). 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 for all outputs (JSON, text and srt). 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_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:

```
{
  "type": "transcription",
  "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 | triple seven five four | 77754 |

| | | | |
|---|---|---|---|
| | 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

Note:

- By default and when speaker diarization is enabled, `speaker` parameter is added per word within the entity, spoken and written form
- When channel diarization is enabled, `channel` parameter is only added on the `results` parent within the entity and not included in spoken and written form


```
"results": [
 {
   "alternatives": [
     {
       "confidence": 0.99,
       "content": "17th of January 2022",
       "language": "en",
       "speaker": "UU"
     }
   ],
   "end_time": 3.14,
   "entity_class": "date",
   "spoken_form": [
     {
       "alternatives": [
         {
           "confidence": 1.0,
           "content": "seventeenth",
           "language": "en",
           "speaker": "UU"
         }
       ],
       "end_time": 1.41,
       "start_time": 0.72,
       "type": "word"
     },
     {
       "alternatives": [
         {
           "confidence": 1.0,
           "content": "of",
           "language": "en",
           "speaker": "UU"
         }
       ],
```

```
      "end_time": 1.53,
      "start_time": 1.41,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "January",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 2.04,
      "start_time": 1.53,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "twenty",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 2.46,
      "start_time": 2.04,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 1.0,
          "content": "twenty",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 2.79,
      "start_time": 2.46,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.97,
          "content": "two",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 3.14,
      "start_time": 2.79,
```

```
        "type": "word"
      }
    ],
    "start_time": 0.72,
    "type": "entity",
    "written_form": [
      {
        "alternatives": [
          {
            "confidence": 0.99,
            "content": "17th",
            "language": "en",
            "speaker": "UU"
          }
        ],
        "end_time": 1.33,
        "start_time": 0.72,
        "type": "word"
      },
      {
        "alternatives": [
          {
            "confidence": 0.99,
            "content": "of",
            "language": "en",
            "speaker": "UU"
          }
        ],
        "end_time": 1.93,
        "start_time": 1.33,
        "type": "word"
      },
      {
        "alternatives": [
          {
            "confidence": 0.99,
            "content": "January",
            "language": "en",
            "speaker": "UU"
          }
        ],
        "end_time": 2.54,
        "start_time": 1.93,
        "type": "word"
      },
      {
        "alternatives": [
          {
            "confidence": 0.99,
            "content": "2022",
            "language": "en",
            "speaker": "UU"
          }
        ],
```

```
          "end_time": 3.14,
          "start_time": 2.54,
          "type": "word"
        }
      ]
    }
  ]
```


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


```
  "results": [
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "content": "17th",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 1.33,
      "start_time": 0.72,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "content": "of",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 1.93,
      "start_time": 1.33,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
          "content": "January",
          "language": "en",
          "speaker": "UU"
        }
      ],
      "end_time": 2.54,
      "start_time": 1.93,
      "type": "word"
    },
    {
      "alternatives": [
        {
          "confidence": 0.99,
```

```
            "content": "2022",
            "language": "en",
            "speaker": "UU"
          }
        ],
        "end_time": 3.14,
        "start_time": 2.54,
        "type": "word"
      }
    ]
}
```


# Batch Container Migration Guide

## Overview

This is a guide for customers who are updating to V8.0.0 or later (October 2020). It documents changes in the batch container, and how you, a customer, may need to reintegrate your batch container with any other systems. It is provided in addition to our standard release notes and documentation pack.

As part of this upgrade, some V1 features that are no longer supported have been completely deprecated, and will cease to work as announced in the v6.2.0 release.

**In all cases, replacements are supported via our V2 input, and are documented in our Speech API Guide.**

The changes below should show no loss of any feature or functionality as a result of the migration.

## Scope

The scope of this document shows:

- What changes you, the customer, must make to use the Speechmatics batch container v8.0.0 if you have been using previous versions of the container
- If you are still using deprecated V1 features, this document will show which ones are no longer supported, and what you must use instead to ensure output
- Examples of our V2 output, and how it differs from our V1 output
- What changes have been made to licensing, and how you, the customer, must license a container prior to using it

The scope of this document excludes

- How to start the Batch Container - this is documented in our quick start guide
- Our Speech API - this is documented in the Speech API guide
- List of software packages used - this is covered in our release and attribution list
- Recommendations for any custom workflows or integrations you have built

## What has changed

### License File

Previously Speechmatics built batch containers with their own license file integrated within the container for each language required by a customer. For simplicity and replicability we have moved to a generic customer-agnostic container for each language, with each customer now receiving a separate license file to use with the container(s) they are licensed for.

Please note: The contents of the license file are confidential. They should be shared on the principles of least privilege. Speechmatics is not responsible for how you handle, store, or share licensing information.

Speechmatics Support will provide you with a new license file. The license is a JSON file called `license.json` and has the following JSON structure:

| Item | Description |
|---|---|
| Customer name | This is your company's name |
| Id | This is internal to Speechmatics |
| Is-Trial | Whether the license is for a trial use of Speechmatics or not |
| Metadata | What Features a container is licensed to use. These can include:<br><br>Speaker Diarization<br><br>Channel Diarization<br><br>Speaker Change<br><br>Batch Container use<br><br>Real-time container use<br><br>Language: any supported language<br><br>Language: A supported individual language (e.g. English) |
| NotValidAfter | The date after which the license expires and can no longer be used to run the container. The date is in ISO format |
| ValidFrom | The date from which this license is valid. |
| Signed Claims Token | A unique reference number used to validate the license file when running the container. Generated by Speechmatics |

The values in this license file will reflect each customer's individual contract arrangement with Speechmatics.

An example license file is below:

```
{
   "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": [
               "MAPBA",
               "LANY"
           ],
```

```
            "isTrial": true,
            "notValidAfter": "2021-01-01",
            "validFrom": "2020-01-01"
        }
    },
    "signedclaimstoken": "example",
}
```

### How this affects you

Previously the batch container was licensed by use of the environment variable `LICENSE_KEY`. This is no longer a valid variable and will not license the product. Instead you may either license the product via the two methods described below:

- **Volume mapping the license file into the container**. Volume map the location of the license file into the container when running transcription jobs, like the Configuration Object. Please see below for an example:

```
docker run -i -v $AUDIO_FILE:/input.audio -v $CONFIG_JSON:/config.json -v /my_license.json:/license.json batch-asr-transcriber-en:8.0.0
```

- **Use the value of the 'signed claims token'** from the license file and pass it as the value of the `LICENSE_TOKEN` variable when running a transcription job. See an example of using LICENSE_TOKEN below:

```
docker run -i -v $AUDIO_FILE:/input.audio -v $CONFIG_JSON:/config.json -e LICENSE_TOKEN='example' batch-asr-transcriber-en:8.0.0
```

If you lose a license file or it is no longer secure, Speechmatics can generate a new one. Please contact Speechmatics support if this is the case.

## V1 Deprecation

In the Speechmatics container you can still process a media file for transcription without use of the V2 configuration object. This will generate our JSON v2 output without any alteration or changes to the text.

From the V8.0.0 release, the configuration file is now the only way by which you can modify the transcription output in the Speechmatics container. If you want to use features such as diarization, punctuation overrides, output locale etc. you must use the configuration object to request these features.

If you already do so, then you do not need to make any changes to how you use the container.

All JSON transcription output will now be in the V2.4 output.

As part of the v7.0.0 release support for V1 features was withdrawn. As part of this release all V1 features have now since been removed. Where applicable, these have been replaced by options within the configuration object. This includes the following:

| V1 Item | Type | Replaced By |
|---|---|---|
| DIARIZE. Enables speaker diarization | environment variable | Use the diarization:speaker parameter within the configuration object |
| DIARISE. Enables speaker diarization | environment variable | Use the diarization:speaker parameter within the configuration object |
| CHANNEL_DIARISATION. enables channel diarization on stereo files | environment variable | Use the diarization:channel parameter within the configuration object |
| CHANNEL_DIARISATION_LABELS. Provides | environemnt | Replaced by the parameter |

| | | |
|---|---|---|
| labels to different speakers when using channel diarization | variable | channel_diarization_labels in the configuration object |
| LICENSE_KEY. used to license the batch container | environment variable | Replaced by LICENSE_TOKEN |
| `/extra_words.txt`. Used as a custom dictionary to generate additional vocabulary objects | text file | Use the additional vocab parameter within the configuration object to generate a custom dictionary |
| `/build_date`. Documents the date the batch container was built by | text file | Replaced by the new licensing file, and no longer needed |
| `/license_days`. How many days the license has to run | text file | Replaced by the new licensing file, and no longer needed |
| | | |

## Changes to Notifications

Notifications are still supported in the batch container as before. There are a few changes in how single and multi-part notifications are generated and encoded, and this is noted below for integration purposes:

- If you request `transcript`, this will now be output in the JSON-V2 format rather than the deprecated V1 JSON format
- If you want to request an empty notification, you **must** specify `contents` to be blank by using `[]`. An example is provided below
- Notifications now have the `charset=utf8` on all transcript types. Ensure that your workflow can support this
- For receiving notifications, `Content-Type` header's used to be set always to `application/octet-stream`. This value now corresponds to actual content of the notification and is `application/json` in case of JSON-v2 content, `text/plain` in case of an SRT contetn, and `application/octet-stream` for TXT content

An example notification configuration that would generate a notification with no contents is shown below. This is a change from the previous version of batch container.

```
{
    "notification_config": [{
        "url": "http://localhost:8080",
        "contents": []
    }]
}
```