API

- API Documentation
- Creating cURL Jobs
- Go Examples
- Javascript Examples
- Ruby Examples
- Python Examples
- Stream videos with HTML-DASH and HLS
- Curl - Check Job Status

API Documentation

Contents

- Parameters
- Cloud Credentials
- Cloud Credentials Input/Output
- Audio Profiles
- Video Profiles
- Supported Containers and Codecs
- Creating Jobs
- Using the API from within Python
- Retrieving a Sent Job

Parameters

input

or not to use this input. This will be an object key if using a cloud provider, or a regular URL otherwise.

Type	Required	Default
string	Yes	

videoProfiles

An array containing at least one video profile.

Type	Required	Default
array	Yes	

audioProfiles

An array containing zero or more audio profiles.

Type	Required	Default
array	Yes	

output_container

Output container to be used. Supported values are ts, mp4, hls, and silverlight .

Type	Required	Default
string	No	mp4

ts_mode_input_audio_source_stream_count

Audio stream count (in TS mode only).

Type	Required	Default
number	No	

audio_output_path

The cloud location where audio should be stored.

Type	Required	Default
string	No	

bucket_output_path

The location, within a cloud bucket, where the output should be stored in segmented mode.

Type	Required	Default

string	No	

master_variant_mode

Whether or not to use separate audio.

Type	Required	Default
boolean	No	false

segmented_output_hls

Whether to use HLS in segmented output.

Type	Required	Default
boolean	No	false

segmented_output_dash

Whether to use DASH in segmented output.

Type	Required	Default
boolean	No	false

segment_length

The segment length in segmented output.

Type	Required	Default
number	No	

video_codec

Output video encoder. Supported values are h.264, mpeg2, and hevc.

Type	Required	Default
string	No	h.264

ip_distance

Target I/P frame distance.

Type	Required	Default
number	No	

gop_length

Target GOP length in seconds.

Type	Required	Default
number	No	1

scte35_passthough

Use SCTE35 passthrough?

Type	Required	Default
boolean	No	false

scte35_pid_remap

PID Remap.

Type	Required	Default
number	No	

create_tar_file

In segmented output, whether to tar the resulting directory and store it in the same bucket specified by the bucket_output_path parameter. Note that this will take up more space in your bucket.

Type	Required	Default
boolean	No	

audio_volume

Target audio volume (0-200).

Type	Required	Default
number	No	100

h264_quality

Output quality. Accepted values are 'good', 'better', and 'best'. Supported values are good, better , and best.

Type	Required	Default
string	No	good

video_aspect_ratio

Output video aspect ratio. Supported values are passthrough, force_4:3, force_16:9, invert_aspect_ratio, rotate_counterclockwise , rotate_clockwise, and rotate_counterclockwise + invert_aspect_ratio.

Type	Required	Default
string	No	passthrough

rotation_blackness

... .

Type	Required	Default
number	No	

closed_captions

Write CC to output?

Type	Required	Default
boolean	No	

force_progressive

Force progressive?

Type	Required	Default
boolean	No	false

keep_fps

Preserve the framerate?

Type	Required	Default
boolean	No	true

logo_url

URL to an image file to use as a watermark.

Type	Required	Default
string	No	

cloud_credentials

See Below

Type	Required	Default
object	Yes	

Cloud Credentials

input

Type	Required	Default
object	Yes	

output

Type	Required	Default
object	Yes	

Cloud Credentials Input/Output

cloud_provider

The cloud provider to use

Supported values are aws, oracle, gcloud, http, file-upload, igolgi-store .

Type	Required	Default
string	Yes	oracle

access_key

Account access key.

Type	Required	Default
string	No	

secret_key

Account secret key.

Type	Required	Default
string	No	

region

The region within which your target bucket is located.

Type	Required	Default
string	No	

---

tenancy_ocid

Tenancy OCID (for oracle only)

Type	Required	Default
string	No	

user_ocid

User OCID (for oracle only)

Type	Required	Default
string	No	

oci_fingerprint

User Fingerprint (for oracle only)

Type	Required	Default
string	No	

project_id

Project ID (for google only)

Type	Required	Default
string	No	

client_email

Client email (for google only)

Type	Required	Default
string	No	

client_id

Client ID (for google only)

Type	Required	Default
string	No	

private_key_id

Private Key ID (for google only)

Type	Required	Default
string	No	

Audio Profiles

audio_codec

Audio codec to be used during transcode. Supported values are aac, ac3 or mp2.

Type	Required	Default
string	No	aac

audio_channels

Number of audio channels on the output. Use 2 for stereo and 6 for 5.1. Supported values are 2 or 6.

Type	Required	Default
string	No	2

audio_bitrate

Output audio bitrate. Supported values are 32, 40, 64, 96, 128, 192, 256, 384, 448 or 768.

Type	Required	Default
string	No	256

primary_audio_downmix_to_stereo

Downmix primary audio from 5.1 to stereo.

Type	Required	Default
boolean	No	

source_stream

The index of the source stream. Supported values are 1 , 2 , 3 , 4 , 5 , 6 , 7 or 8.

Type	Required	Default
string	No	

dolby_digital_dialnorm

The dialnorm. Values range from 0 to 31 inclusive. Use 0 for passthrough.

Type	Required	Default
number	No	

Video Profiles

output

The location, including the file name, where the output is to be sent.

Type	Required	Default
string	No	

width

The output width in pixels.

Type	Required	Default
number	Yes	

height

The output height in pixels

Type	Required	Default
number	Yes	

video_bitrate_mode

Output video bitrate mode. Supported values are cbr or vbr.

Type	Required	Default
string	No	

video_bitrate

The video bitrate in kbps.

Type	Required	Default
number	Yes	

video_framerate

The video framerate. Supported values are `1x`, `1/2x` or `2x`.

Type	Required	Default
string	No	`1x`

audio_profiles

Type	Required	Default
string	No	

Supported Containers and Codecs

- Input Containers: MXF, MPEG2-TS and MP4
- Output Containers: MPEG2-TS, MP4
- Input Codecs: MPEG2, ProRes, XDCAM, H.264, HEVC for video; AC3, MPEG2, AAC for audio
- Output Codecs: MPEG2, H.264 and HEVC for video; Passthrough, AAC-LC and AAC 5.1 for audio, AC3 stereo and 5.1 for audio

Creating Jobs

Using the API from within Python

Retrieving a Sent Job

Job Status: {'job_id': '4765', 'tcode_progress': '100', 'state': 'ready', 'tcode_time': '57.89', 'tcode_speed': '0.0', 'job_completed': True, 'xfer_speed': '337.88', 'xfer_time': '0.67', 'xfer_progress': '100'}

Job Config: {'input': 'https://streamengine.igolgi.com/uploads/659d9ac25dc7bf7b1a7c487a971a98e8', 'videoProfiles': [{'output': 'oci://yzwhip2gci7s/igolgi-public/80d8cc2a_bdab_442b_bc19_21fe00267a34.mp4', 'width': 0, 'height': 0, 'video_bitrate': 6000, 'video_framerate': '1x'}], 'audioProfiles': [{'audio_codec': 'aac', 'audio_channels': 2, 'audio_bitrate': 128, 'primary_audio_downmix_to_stereo': False, 'source_stream': 1}], 'output_container': 'mp4', 'audio_output_path': 'oci://yzwhip2gci7s/igolgi-public/96dbafd7_bf4f_45a8_bd06_627a4684e45b.mp4', 'separate_audio': True, 'segmented_output_hls': False, 'segmented_output_dash': False, 'video_codec': 'hevc', 'ip_distance': 3, 'gop_length': 2, 'scte35_passthough': False, 'scte35_pid_remap': -1, 'create_tar_file': False, 'audio_volume': 100, 'h264_quality': 'good', 'video_aspect_ratio': 'rotate_counterclockwise + invert_aspect_ratio', 'rotation_blackness': 0, 'closed_captions': False, 'picture_transform': 'none', 'logo_url': None}

Creating cURL Jobs

Creating Jobs

To create a job, use your API key along with a JSON payload matching the schema described in the tables above, and make an HTTP GET request like the following:

```
curl https://streamengine.igolgi.com/api/v0/job \
  -k \
  --request POST \
  -u 8e6f7a5bf90821a175338ce08be4b7ca:a \
  --header 'Content-Type: application/json' \
  -d '{
  "videoProfiles": [
    {
      "output": "",
      "width": 1024,
      "height": 768,
      "video_bitrate": 3500,
      "video_bitrate_mode": "cbr",
      "video_framerate": "1x"
    }
  ],
  "audioProfiles": [
    {
      "audio_bitrate": 256,
      "audio_channels": 2,
      "audio_codec": "aac"
    }
  ],
  "output_container": "mp4",
  "video_codec": "h.264",
  "input": "oci://yzwhip2gci7s/jey_test_bucket2/test_dir/moana_30_seconds.ts",
  "cloud_credentials": {
    "input": {
      "cloud_provider": "oracle",
      "access_key": "...",
      "secret_key": "...",
      "region": "ca-toronto-1"
    }
  }
}'
```

Retrieving a Sent Job

To retrieve a job, use your API key to make a GET request to the `/api/v0/job/id` endpoint, replacing `id` with the id of the job you wish to retrieve:

```
curl https://streamengine.igolgi.com/api/v0/job/1 \
  -u 8e6f7a5bf90821a175338ce08be4b7ca:
```

The response will look something like this:

```
{
   "status": {
      "tcode_progress": "0",
      "state": "ready",
      ... # additional fields
   },
   "config": {
      ... # your job configuration
   },
}
```

This allows you to retrieve the original configuration for your job, as well as check on its progress using the `tcode_progress` field.

Go Examples

This is an example in Go, please modify it to fit your environment, be sure to change the video input and output paths as well as the API_KEY and AWS_ACCESS and AWS_SECRET keys.

```
package main

import (
	"bytes"
	"encoding/json"
	"fmt"
	"io/ioutil"
	"net/http"
)

func main() {
	url := "https://streamengine.igolgi.com/api/v0/job"

	// Authentication credentials
	username := "INSERT_YOUR_API_KEY"
	password := "a"

	// Data to be sent in the POST request
	data := map[string]interface{}{
		"videoProfiles": []map[string]interface{}{
			{
				"output":              "s3://igolgi-se-out/go-testcwaudio3f.mp4",
				"width":               1920,
				"height":              1080,
				"video_bitrate_mode":  "cbr",
				"video_bitrate":       2301,
				"video_format":        "progressive",
				"video_framerate":     "1x",
				"audio_profiles":      "1",
			},
		},
		"audioProfiles": []map[string]interface{}{
```

```
		{
			"audio_bitrate":   256,
			"audio_channels":  2,
			"audio_codec":     "aac",
			"source_stream":   1,
		},
	},
	"master_variant_mode": false,
	"output_container":    "mp4",
	"video_codec":         "h.264",
	"input":               "s3://igolgi-se-in/GOPR8297-2.7K-30-fps.MP4",
	"auto_detelecine_flag": false,
	"cloud_credentials": map[string]interface{}{
		"input": map[string]string{
			"cloud_provider": "aws",
			"access_key":     "INSERT_YOUR_AWS_ACCESS_KEY",
			"secret_key":     "INSERT_YOUR_AWS_SECRET_KEY",
			"region":         "us-east-2",
		},
		"output": map[string]string{
			"cloud_provider": "aws",
			"access_key":     "INSERT_YOUR_AWS_ACCESS_KEY",
			"secret_key":     "INSERT_YOUR_AWS_SECRET_KEY",
			"region":         "us-east-2",
		},
	},
}

jsonData, err := json.Marshal(data)
if err != nil {
	fmt.Println("Error marshalling JSON:", err)
	return
}

req, err := http.NewRequest("POST", url, bytes.NewBuffer(jsonData))
if err != nil {
	fmt.Println("Error creating request:", err)
	return
}
```

```
	req.Header.Set("Content-Type", "application/json")
	req.SetBasicAuth(username, password)

	client := &http.Client{}
	resp, err := client.Do(req)
	if err != nil {
		fmt.Println("Error sending request:", err)
		return
	}
	defer resp.Body.Close()

	body, err := ioutil.ReadAll(resp.Body)
	if err != nil {
		fmt.Println("Error reading response:", err)
		return
	}

	fmt.Println("Response:", string(body))
}
```

save the script above as se_api_aws.go or similar

run
  go run se_api_aws.go

it will give print something like:
Response: {"_auto_generated_id_":**13455**,"cloud":"aws" ....

Remember your job number above and use it with the following script to check the status.

```
package main

import (
	"encoding/base64"
	"encoding/json"
	"fmt"
	"io/ioutil"
	"net/http"
	"os"
)
```

```
func main() {
    if len(os.Args) < 2 {
        fmt.Println("Please provide a job ID as an argument.")
        os.Exit(1)
    }

    jobID := os.Args[1]
    url := fmt.Sprintf("https://streamengine.igolgi.com/api/v0/job/%s", jobID)
    apiKey := "INSERT_YOUR_API_KEY" // Replace with your actual API key

    req, err := http.NewRequest("GET", url, nil)
    if err != nil {
        fmt.Println("Error creating request:", err)
        return
    }

    // Set Authorization header
    authHeader := base64.StdEncoding.EncodeToString([]byte(apiKey + ":"))
    req.Header.Set("Authorization", "Basic "+authHeader)

    client := &http.Client{}
    resp, err := client.Do(req)
    if err != nil {
        fmt.Println("Error sending request:", err)
        return
    }
    defer resp.Body.Close()

    body, err := ioutil.ReadAll(resp.Body)
    if err != nil {
        fmt.Println("Error reading response:", err)
        return
    }

    if resp.StatusCode == 200 {
        var data map[string]interface{}
        err = json.Unmarshal(body, &data)
        if err != nil {
            fmt.Println("Error parsing JSON:", err)
            return
```

```
    }

    fmt.Println("Job Status:")
    fmt.Println("------------------------")
    status := data["status"].(map[string]interface{})
    fmt.Printf("Job ID: %s\n", status["job_id"])
    fmt.Printf("State: %s\n", status["state"])
    fmt.Println("Progress:")
    fmt.Printf("  Transcode: %v%%\n", status["tcode_progress"])
    fmt.Printf("  Transfer: %v%%\n", status["xfer_progress"])
    fmt.Println("Time:")
    fmt.Printf("  Transcode: %v seconds\n", status["tcode_time"])
    fmt.Printf("  Transfer: %v seconds\n", status["xfer_time"])
    fmt.Println("Speed:")
    fmt.Printf("  Transcode: %v x\n", status["tcode_speed"])
    fmt.Printf("  Transfer: %v MB/s\n", status["xfer_speed"])
    fmt.Printf("Job Completed: %v\n", status["job_completed"])

    fmt.Println("\nJob Config:")
    fmt.Println("------------------------")
    config, err := json.MarshalIndent(data["config"], "", "  ")
    if err != nil {
        fmt.Println("Error formatting config:", err)
    } else {
        fmt.Println(string(config))
    }
} else {
    fmt.Printf("Failed to retrieve job. Status code: %d\n", resp.StatusCode)
    fmt.Printf("Response: %s\n", string(body))
}
}
```

Save the script above as se-api-status.go or similar.

Run
  go run se-api-status.go **13455**

You'll see a result similar to the one below. The job ID of status will be different than the submit job id. You can rerun the status until you see Job Completed: true.

Job Status:

Job ID: 4920

State: ready

Progress:

  Transcode: 0%

  Transfer: 0%

Time:

  Transcode: <nil> seconds

  Transfer: 78460.45 seconds

Speed:

  Transcode: -1 x

  Transfer: -1.0 MB/s

Job Completed: false

Job Config:

```
{
  "audioProfiles": [
    {
      "audio_bitrate": 256,
      "audio_channels": 2,
      "audio_codec": "aac",
      "source_stream": 1
    }
  ],
  "audio_volume": 100,
  "gop_length": 1,
  "h264_quality": "good",
  "input": "s3://igolgi-se-in/GOPR8297-2.7K-30-fps.MP4",
  "output_container": "mp4",
  "picture_transform": "none",
  "scte35_passthough": false,
  "segmented_output_dash": false,
  "segmented_output_hls": false,
  "separate_audio": false,
  "videoProfiles": [
    {
      "audio_profiles": "1",
      "height": 1080,
```

```
      "output": "s3://igolgi-se-out/go-testcwaudio3f.mp4",
      "video_bitrate": 2301,
      "video_bitrate_mode": "cbr",
      "video_framerate": "1x",
      "width": 1920
    }
  ],
  "video_aspect_ratio": "passthrough",
  "video_codec": "h.264"
}
```


On linux/Ubuntu if you have never used go.

You can run:

$ sudo apt update
$ sudo apt install golang-go

Javascript Examples

The following code sample was written in java using the fetch method.

It requires an API_KEY and FILE_NUMBER_OF_UPLOADED_VIDEO

Depending on your operating system you will need node.js loaded https://nodejs.org/en
and this code can be run:

node se-localfile.mjs
node --experimental-modules se-localfile.mjs

se-localfile.mjs

```
import fetch from 'node-fetch';

const url = 'https://streamengine.igolgi.com/api/v0/job';
const username = 'API_KEY';
const password = 'a';

const headers = {
  'Authorization': 'Basic ' + Buffer.from(username + ":" + password).toString('base64'),
  'Content-Type': 'application/json'
};

const data = {
  videoProfiles: [
    {
      width: 960,
      height: 544,
      video_bitrate: 2500,
      video_framerate: "1x",
      audio_profiles: "0"
    }
  ],
  separate_audio: false,
```

```
segmented_output_dash: false,
output_container: "mp4",
picture_transform: "none",
logo_url: null,
audio_volume: 100,
closed_captions: false,
create_tar_file: false,
gop_length: 2,
h264_quality: "good",
scte35_pid_remap: -1,
video_aspect_ratio: "rotate_counterclockwise + invert_aspect_ratio",
rotation_blackness: 0,
scte35_passthough: false,
segmented_output_hls: false,
ip_distance: 2,
audioProfiles: [
  {
    audio_codec: "aac",
    audio_channels: 2,
    audio_bitrate: 256,
    primary_audio_downmix_to_stereo: false,
    source_stream: 1
  }
],
input: "https://streamengine.igolgi.com/uploads/FILE_NUMBER_OF_UPLOADED_VIDEO",
cloud_credentials: {
  input: {
    cloud_provider: "file-upload",
    access_key: null,
    secret_key: null,
    region: "ca-toronto-1",
    tenancy_ocid: null,
    user_ocid: null,
    oci_fingerprint: null
  },
  output: {
    cloud_provider: "igolgi-store",
    access_key: null,
    secret_key: null,
    region: "ca-toronto-1",
```

```
      tenancy_ocid: null,
      user_ocid: null,
      oci_fingerprint: null
    }
  },
  master_variant_mode: false,
  video_codec: "h.264"
};

fetch(url, {
  method: 'POST',
  headers: headers,
  body: JSON.stringify(data)
})
.then(response => response.json())
.then(data => console.log(data))
.catch(error => console.error('Error:', error));
```

---

se-job-status.mjs

`$node se-job-status.mjs  job-id`

This sample script will run every 10 seconds to see if a job is still running, when its finished it will display finished, feel free to modify this.

```
import fetch from 'node-fetch';

const checkJobStatus = async (jobId) => {
  const url = `https://streamengine.igolgi.com/api/v0/job/${jobId}`;  // Adjust this URL as needed
  const username = 'API_KEY';
  const password = 'a';

  const headers = {
    'Authorization': 'Basic ' + Buffer.from(username + ":" + password).toString('base64'),
```

```
    'Content-Type': 'application/json'
  };

  try {
    const response = await fetch(url, {
      method: 'GET',
      headers: headers
    });

    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data = await response.json();
    console.log('Job status:', data);
    return data;
  } catch (error) {
    console.error('Error checking job status:', error);
  }
};

const checkUntilComplete = async (jobId) => {
  let status;
  do {
    const result = await checkJobStatus(jobId);
    status = result.status;  // Adjust this based on the actual API response structure
    if (status !== 'completed') {
      console.log('Job still in progress. Waiting 10 seconds before checking again...');
      await new Promise(resolve => setTimeout(resolve, 10000));  // Wait for 10 seconds
    }
  } while (status !== 'completed');
  console.log('Job completed!');
};

// Get job ID from command line argument
const jobId = process.argv[2];

if (!jobId) {
  console.error('Please provide a job ID as a command-line argument.');
  console.error('Usage: node se-job-status.mjs <job_id>');
```

```
  process.exit(1);
}

console.log(`Checking status for job ID: ${jobId}`);
checkUntilComplete(jobId);
```

Ruby Examples

Feel free to modify the following script for your environment. To process the video, this script below assumes your job is on aws in an S3 bucket for input and output.

```
require 'net/http'
require 'uri'
require 'json'

url = URI.parse("https://streamengine.igolgi.com/api/v0/job")

headers = {
  'Content-Type' => 'application/json'
}

# Authentication credentials (username and password)
auth = ['INSERT_YOUR_API_KEY', 'a']

# Data to be sent in the POST request
data = {
  videoProfiles: [
    {
      output: 's3://igolgi-se-out/rb-testcwaudio3e.mp4',
      width: 1920,
      height: 1080,
      video_bitrate_mode: 'cbr',
      video_bitrate: 2400,
      video_format: 'progressive',
      video_framerate: '1x',
      audio_profiles: '1'
    }
  ],
  audioProfiles: [
    {
      audio_bitrate: 256,
      audio_channels: 2,
      audio_codec: 'aac',
```

```
      source_stream: 1
    }
  ],
  master_variant_mode: false,
  output_container: "mp4",
  video_codec: "h.264",
  input: "s3://igolgi-se-in/GOPR8297-2.7K-30-fps.MP4",
  auto_detelecine_flag: false,
  cloud_credentials: {
    input: {
      cloud_provider: "aws",
      access_key: "INSERT_YOUR_AWS_ACCESS_KEY",
      secret_key: "INSERT_YOUR_AWS_SECRET_KEY",
      region: "us-east-2"
    },
    output: {
      cloud_provider: "aws",
      access_key: "INSERT_YOUR_AWS_ACCESS_KEY",
      secret_key: "INSERT_YOUR_AWS_SECRET_KEY",
      region: "us-east-2"
    }
  }
}

http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true
http.verify_mode = OpenSSL::SSL::VERIFY_NONE # Not recommended for production

request = Net::HTTP::Post.new(url.path, headers)
request.basic_auth(auth[0], auth[1])
request.body = data.to_json

response = http.request(request)

# Print the response content
puts response.body
```

Save the file as **se-api-aws.rb** or similar.

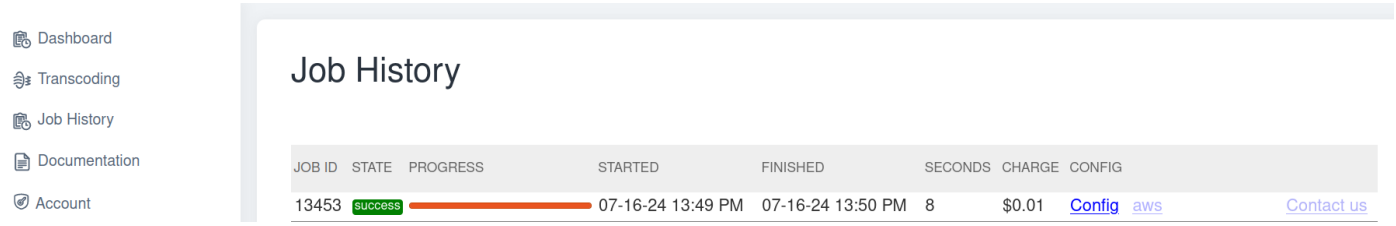
Run:
ruby se-api-aws.rb

You may notice an error like this:
...error ... message: "The video bitrate for this profile and codec should set at a minimum of 2300."...

The error above means the video_bitrate variable in the code should be higher than 2300, it was already changed in the code above, but used as an example of the messages you might see.

When it runs correctly you might see something similar to this:
{"_auto_generated_id_":**13453**,..."updated_at":"2024-07-16T18:49:57.000Z"}

The Id number can also be found on the Job History page.



If you want to see the status of the job from ruby you can use this code:

```
require 'net/http'
require 'uri'
require 'json'

# Check if an argument was provided
if ARGV.length > 0
  external_var = ARGV[0]
else
  puts "Please provide a job ID as an argument."
  exit
end

# Define the URL and API key
url = URI.parse("https://streamengine.igolgi.com/api/v0/job/#{external_var}")
api_key = "1a944b985b09c14b5001ba5cc1a1f585"

# Set up the HTTP request
http = Net::HTTP.new(url.host, url.port)
http.use_ssl = true

request = Net::HTTP::Get.new(url.request_uri)
request.basic_auth(api_key, '')
```

```
# Make the GET request
response = http.request(request)

# Check for a successful response
if response.code == '200'
  puts "Job Status:"
  puts "------------------------"
  puts "Job ID: #{data['status']['job_id']}"
  puts "State: #{data['status']['state']}"
  puts "Progress:"
  puts "  Transcode: #{data['status']['tcode_progress']}%"
  puts "  Transfer: #{data['status']['xfer_progress']}%"
  puts "Time:"
  puts "  Transcode: #{data['status']['tcode_time']} seconds"
  puts "  Transfer: #{data['status']['xfer_time']} seconds"
  puts "Speed:"
  puts "  Transcode: #{data['status']['tcode_speed']} x"
  puts "  Transfer: #{data['status']['xfer_speed']} MB/s"
  puts "Job Completed: #{data['status']['job_completed']}"

  puts "\nJob Config:"
  puts "------------------------"
  puts JSON.pretty_generate(data['config'])
else
  puts "Failed to retrieve job. Status code: #{response.code}"
  puts "Response: #{response.body}"
end
```

Save the file above as **se_api_status.rb** or similar

Run
ruby se_api_status.rb 13453 where 13453 is the job number

```
Job Status:
------------------------
Job ID: 4918
State: ready
Progress:
  Transcode: 100%
  Transfer: 0%
```

Time:
  Transcode: 19.06 seconds
  Transfer: 28.29 seconds
Speed:
  Transcode: 0.0 x
  Transfer: 0.0 MB/s
Job Completed: true

Job Config:

```
{
  "input": "s3://igolgi-se-in/GOPR8297-2.7K-30-fps.MP4",
  "videoProfiles": [
    {
      "output": "s3://igolgi-se-out/py2-testcwaudio3e.mp4",
      "width": 1920,
      "height": 1080,
      "video_bitrate_mode": "cbr",
      "video_bitrate": 2400,
      "video_framerate": "1x",
      "audio_profiles": "1"
    }
  ],
  "audioProfiles": [
    {
      "audio_codec": "aac",
      "audio_channels": 2,
      "audio_bitrate": 256,
      "source_stream": 1
    }
  ],
  "output_container": "mp4",
  "separate_audio": false,
  "segmented_output_hls": false,
  "segmented_output_dash": false,
  "video_codec": "h.264",
  "gop_length": 1,
  "scte35_passthough": false,
  "audio_volume": 100,
  "h264_quality": "good",
```

```
  "video_aspect_ratio": "passthrough",
  "picture_transform": "none"
}
```

Python Examples

The following example uses video files on AWS in an S3 bucket and writes files out to a different AWS S3 Bucket.

```
import requests

url = "https://streamengine.igolgi.com/api/v0/job"

headers = {
   'Content-Type': 'application/json',
}

# Authentication credentials (username and password)
auth = ('INSERT_YOUR_API_KEY', 'a')

# Data to be sent in the POST request
data = {
   'videoProfiles': [
      {
         'output': 's3://igolgi-se-out/py2-testcwaudio3d.mp4',
         'width': 1920,
         'height': 1080,
         'video_bitrate_mode': 'cbr',
         'video_bitrate': 2250,  # there are minimum bitrates see tables
         'video_format': 'progressive',
         'video_framerate': '1x',
         'audio_profiles': '1'   # required if you want audio 0 - 7
      }],
   'audioProfiles': [
      {
         'audio_bitrate': 256,
         'audio_channels': 2,
         'audio_codec': 'aac',
         'source_stream': 1   # required if you want audio 1 - 8
      }],
```

```
    "master_variant_mode": False,
    "output_container": "mp4",
    "video_codec": "h.264",
    "input": "s3://igolgi-se-in/GOPR8297-2.7K-30-fps.MP4",
    "auto_detelecine_flag": False,
    "cloud_credentials": {
      "input": {
        "cloud_provider": "aws",
        "access_key": "INSERT_YOUR_AWS_ACCESS_KEY",
        "secret_key": "INSERT_YOUR_AWS_SECRET_KEY",
        "region": "us-east-2"
      },
       "output": {
        "cloud_provider": "aws",
        "access_key": "INSERT_YOUR_AWS_ACCESS_KEY",
        "secret_key": "INSERT_YOUR_AWS_SECRET_KEY",
        "region": "us-east-2"
      }
    }
}

# Disable SSL verification (not recommended for production)
response = requests.post(url, json=data, headers=headers, auth=auth)

# Print the response content
print(response.content.decode('utf-8'))
```

```
import requests
import sys
import json
from requests.auth import HTTPBasicAuth

if len(sys.argv) > 1:
    external_var = sys.argv[1]
else:
    print("Please provide a job ID as an argument.")
    sys.exit(1)
```

```
# Define the URL and API key
url = f"https://streamengine.igolgi.com/api/v0/job/{external_var}"
api_key = "INSERT_YOUR_API_KEY"

# Make the GET request
response = requests.get(url, auth=HTTPBasicAuth(api_key, ''))

# Check for a successful response
if response.status_code == 200:
    data = response.json()
    status = data.get("status", {})
    config = data.get("config", {})

    print("Job Status:")
    print("------------------------")
    print(f"Job ID: {status.get('job_id')}")
    print(f"State: {status.get('state')}")
    print("Progress:")
    print(f"  Transcode: {status.get('tcode_progress')}%")
    print(f"  Transfer: {status.get('xfer_progress')}%")
    print("Time:")
    print(f"  Transcode: {status.get('tcode_time')} seconds")
    print(f"  Transfer: {status.get('xfer_time')} seconds")
    print("Speed:")
    print(f"  Transcode: {status.get('tcode_speed')} x")
    print(f"  Transfer: {status.get('xfer_speed')} MB/s")
    print(f"Job Completed: {status.get('job_completed')}")

    print("\nJob Config:")
    print("------------------------")
    print(json.dumps(config, indent=2))

else:
    print(f"Failed to retrieve job. Status code: {response.status_code}")
    print("Response:", response.text)
```

```
Job Status:
------------------------
```

Job ID: 4920
State: ready
Progress:
  Transcode: 100%
  Transfer: 0%
Time:
  Transcode: 19.94 seconds
  Transfer: 27.87 seconds
Speed:
  Transcode: 0.0 x
  Transfer: 0.0 MB/s
Job Completed: True

Job Config:

```
------------------------
{
  "input": "s3://igolgi-se-in/GOPR8297-2.7K-30-fps.MP4",
  "videoProfiles": [
    {
      "output": "s3://igolgi-se-out/go-testcwaudio3f.mp4",
      "width": 1920,
      "height": 1080,
      "video_bitrate_mode": "cbr",
      "video_bitrate": 2301,
      "video_framerate": "1x",
      "audio_profiles": "1"
    }
  ],
  "audioProfiles": [
    {
      "audio_codec": "aac",
      "audio_channels": 2,
      "audio_bitrate": 256,
      "source_stream": 1
    }
  ],
  "output_container": "mp4",
  "separate_audio": false,
  "segmented_output_hls": false,
  "segmented_output_dash": false,
```

```
  "video_codec": "h.264",
  "gop_length": 1,
  "scte35_passthough": false,
  "audio_volume": 100,
  "h264_quality": "good",
  "video_aspect_ratio": "passthrough",
  "picture_transform": "none"
}
```

Stream videos with HTML-DASH and HLS

Why would I want to use hls or dash?
Dash and Hls are great for streaming large files and give the user the ability to jump around in the video, fast forward, rewind or jump to an index. In the past you needed expensive streaming servers and serve videos using rtsp: or mms:

In StreamEngine convert a file you want to stream from a website to mp4-hls, mp4-dash or mp4-hls+dash. This will create a tar or tar.gz file
DASH - Dynamic Adaptive Streaming over HTTP

then copy the file to your webserver and name it playlist.m3u8 or modify the <source src = "playlist.m3u8" ... line below.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>HLS-DASH Player</title>
    <link href="https://vjs.zencdn.net/7.20.3/video-js.min.css" rel="stylesheet">
    <script src="https://vjs.zencdn.net/7.20.3/video.min.js"></script>
</head>
<body>
    <video-js id="my-video" class="video-js vjs-default-skin vjs-big-play-centered" controls preload="auto" width="640" height="360">
        <source src="playlist.m3u8" type="application/x-mpegURL">
    </video-js>

    <script>
        var player = videojs('my-video');
        player.play();
    </script>
```

```
</body>
</html>
```

This HTML file does the following:

1. It includes the video.js CSS and JavaScript files from a CDN.
2. It creates a video player using the `video-js` tag.
3. It sets the source of the video to your `playlist.m3u8` file.
4. It initializes the video.js player and attempts to start playback.

To use this:

1. Save this HTML code to a file (e.g., `player.html`) on your web server.
2. Make sure your `playlist.m3u8` file is in the same directory as the HTML file, or update the `src` attribute with the correct path to your playlist file.
3. Access the HTML file through your web server (e.g., `https://yourserver.com/player.html`).

Note:

- Ensure that your web server is correctly configured to serve the `.m3u8` file with the correct MIME type (`application/x-mpegURL`).
- The player's size is set to 640x360 pixels. You can adjust these values as needed.
- This example uses video.js version 7.20.3. You may want to check for the latest version and update the URLs accordingly.

Curl - Check Job Status

When you submit a curl job you should get a response back from the StreamEngine server with the job id on it.

Curl - Linux and Windows

curl -u API_KEY:a -H "Content-Type: application/json"
https://streamengine.igolgi.com/api/v0/job/<JOB_ID>

replace API_KEY with your StreamEngine API Key

replace <JOB_ID>

If you only want to see job status and have jq installed on linux you can run

curl -u API_KEY:a -H "Content-Type: application/json"
https://streamengine.igolgi.com/api/v0/job/<JOB_ID> | jq ".status"

```
$ curl -u API_KEY:a -H "Content-Type: application/json" https://streamengine.igolgi.com/api/v0/job/<JOB_ID> | jq ".status"

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1079  100  1079    0     0   9989      0 --:--:-- --:--:-- --:--:-- 10084
{
  "job_id": "5087",
  "tcode_progress": "100",
  "state": "ready",
  "tcode_time": "36.84",
  "tcode_speed": "0.0",
  "job_completed": true,
  "xfer_speed": "0.0",
  "xfer_time": "86398.93",
  "xfer_progress": "100"
}
```

Note the job_id above (backend) will be different than is the <JOB_ID> submitted which is the id of the frontend,

or

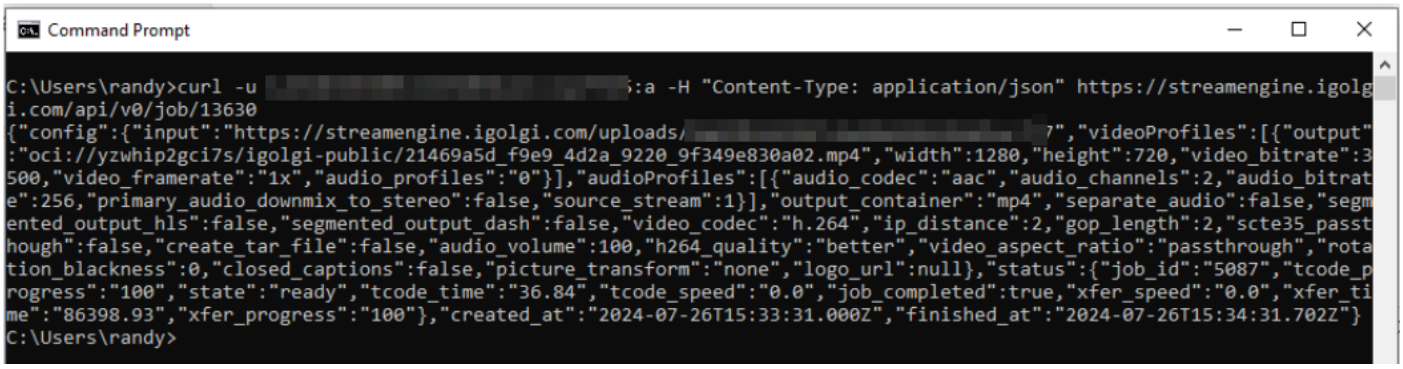
```
$ curl -u API_KEY:a -H "Content-Type: application/json" https://streamengine.igolgi.com/api/v0/job/<JOB_ID> | jq ".status.job_completed"

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  1079  100  1079    0     0   9099      0 --:--:-- --:--:-- --:--:--  9144

true
```

jq is available for Windows, use at your own risk, it's a separate install, AMD64 executable should work with intel.

Windows cmd, note API_KEY blurred out.



On Windows with PowerShell. PowerShell is a little different curl works but -u option does not so we will need to write a program.

```
param(
    [Parameter(Mandatory=$true)]
    [string]$JobId
)

$url = "https://streamengine.igolgi.com/api/v0/job/$JobId"
$username = "API_KEY"
$password = "a"

$base64AuthInfo = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes(("{0}:{1}" -f $username,$password)))
```

```
$headers = @{
    Authorization = "Basic $base64AuthInfo"
    "Content-Type" = "application/json"
}

try {
    $response = Invoke-RestMethod -Uri $url -Headers $headers -Method Get

    Write-Host "Status for Job ID: $JobId"
    Write-Host "Full status:"
    $response.status | ConvertTo-Json

    Write-Host "`nKey status fields:"
    Write-Host "State: $($response.status.state)"
    Write-Host "Job Completed: $($response.status.job_completed)"
    Write-Host "Tcode Progress: $($response.status.tcode_progress)%"
    Write-Host "Xfer Progress: $($response.status.xfer_progress)%"
}
catch {
    Write-Host "Error checking job status: $_"
}
```

save it to a file for example CheckJobStatus.ps1

Then you can run the script above.

.\CheckJobStatus.ps1 -JobID <JOB_ID>

Linux shell script if you want to store your API key an reuse it with a script.

```
#!/bin/bash

JOB_ID=$1

if [ -z "$JOB_ID" ]; then
    echo "Please provide a job ID as an argument."
    echo "Usage: ./check_job_status.sh <job_id>"
```

```
    exit 1
fi

echo "Checking status for job ID: $JOB_ID"

while true; do
    response=$(curl -s -u API_KEY:a -H "Content-Type: application/json"
https://streamengine.igolgi.com/api/v0/job/$JOB_ID)
    status=$(echo $response | jq -r '.status')

    echo "Job status: $status"

    if [ "$status" = "completed" ]; then
        echo "Job completed!"
        break
    else
        echo "Job still in progress. Waiting 10 seconds before checking again..."
        sleep 10
    fi
done
```

Curl - Windows PowerShell

Here is another example on Windows in PowerShell

.\CheckJobStatus.ps1 -JobId <Your-Job-ID>

```
param(
    [Parameter(Mandatory=$true)]
    [string]$JobId
)

$url = "https://streamengine.igolgi.com/api/v0/job/$JobId"
$username = "API_KEY"
$password = "a"

$base64AuthInfo = [Convert]::ToBase64String([Text.Encoding]::ASCII.GetBytes(("{0}:{1}" -f
$username,$password)))

$headers = @{
    Authorization = "Basic $base64AuthInfo"
```

```
    "Content-Type" = "application/json"
}

function Check-JobStatus {
    try {
        $response = Invoke-RestMethod -Uri $url -Headers $headers -Method Get
        return $response
    }
    catch {
        Write-Host "Error checking job status: $_"
        return $null
    }
}

Write-Host "Checking status for job ID: $JobId"

do {
    $result = Check-JobStatus
    if ($result -ne $null) {
        $status = $result.status  # Adjust this if the API response structure is different
        Write-Host "Job status: $status"

        if ($status -ne "completed") {
            Write-Host "Job still in progress. Waiting 10 seconds before checking again..."
            Start-Sleep -Seconds 10
        }
    }
    else {
        Write-Host "Failed to get job status. Retrying in 10 seconds..."
        Start-Sleep -Seconds 10
    }
} while ($status -ne "completed")

Write-Host "Job completed!"
```