Vantrix Corporation
VTA Developer Guide

Version: 61

Date: September 7, 2016

Vantrix Corporation
1425 Rene-Levesque West, Suite 1200
Montreal, Quebec H3G 1T7 CANADA
Tel: +1 (514) 866-1717
Fax: +1 (514) 866-6868
http://www.vantrix.com/

Contents

Introduction

This *VTA Developer Guide* exposes the most commonly used transcoding tasks and provides recipes to help you formulate the job object to be sent to the VTA service using the REST API. The aim is to include most transcoding cases and explanations about how to submit a proper job definition. This guide provides mainly complete job samples apart from a small section about preset usage. Almost every recipe demonstrated can be used to create a preset where convenient.

Please refer to the *VTA API Specification* for more details about the job model and the preset concepts.

1 Basics

In this section, we cover the API usage basics and provide a few samples. Please refer to the *VTA API Specification* for additional details.

1.1 Using the API

All interactions with the API are done using the REST model over HTTP transport. The data blocks are coded using the JSON format often used for such APIs. The calls require the "*Content-Type: application/json*" header when sending a JSON block (create, update, patch), and require the "*x-vta-key: your-key-here*" header with your key. An invalid or expired key will result in a 403 Forbidden status code.

```
POST http://host:12200/vta/v1/jobs/transcode
Content-Type: application/json
X-Vta-Key: 12345
{
  JSON DATA
}
```

1.2 Validations and 400 status code

The service validates any JSON block that it receives for creation or update. Every error found is returned in a JSON response message with an HTTP status 400.

For instance, trying to create a transcoding job without a file URL in the input would result in:

```
{
  "type": "validation",
  "message": "Validation error. Request cannot be completed.",
  "errorCount": 1,
  "failedValidations": [
    "Field [job.input.url] with value of [null] -> may not be null"
  ]
}
```

1.3 Searching and Filtering

The API provides powerful searching and filtering capabilities. Any field in the *preset* or *job* object is searchable. Criteria can be combined and multiple values can be specified. For instance, you can request all jobs having a *queued* or *processing* status producing an *HLS* container format with an *HD* definition created less than *10 minutes* ago. All these are criteria that can be specified in corresponding parameters when requesting jobs. The same applies for presets. Please refer to the *VTA API Specification* for additional details.

Search parameters can be combined and will be evaluated as a logical AND. Thus one could query all jobs that are of status processing AND assigned to pool id “live”, for instance. Such a query would look like the following:

```
http://host:12200/vta/v1/presets?status=processing&poolId=live
```

Here are a few other samples.

1.3.1 Listing system presets

```
http://host:12200/vta/v1/presets?scope=system
```

1.3.2 Finding a job by name

```
http://host:12200/vta/v1/jobs/transcoding?name=ChannelName
```

1.3.3 Finding all incomplete jobs

```
http://host:12200/vta/v1/jobs/transcode?status=submitted,queued,
processing,paused
```

1.3.4 Combining criteria: all processing, live

```
http://host:12200/vta/v1/jobs/transcode?input.type=live&status=proc
essing
```

1.4 Paging and Sorting

The API also provides a powerful mechanism for paging responses and sorting a result set. It uses the same searching and filtering field-naming scheme to designate a specific field in a series of embedded JSON objects. Then you can request to retrieve the result set using an ASCending or DESCending order.

The use cases supported are:

- Getting all jobs with default paging (no parameters)
  VTA will rely on its internal defaults and will return page 0 with the default number of elements. Configuration properties manage these defaults.
- Getting a specific page number with a specific size
  For instance, the client can ask for page 2 with 10 jobs. In this case VTA will return jobs #10-19.
- Trying to get all the jobs in a single call
  For instance, the client asks for page 0 with a million jobs thinking he will get them all. In this situation, VTA will rely on its ceiling value of 2000 elements (you may double-check the value in the current release as it could have changed over time) and return page 0 with 2000 jobs.
- Sorting the result on a single field ascending or descending
- Sorting on multiple fields

1.4.1 Response headers

The REST call will return four headers in the response to describe the paging details.

Here’s a sample response with the four headers:

```
GET http://host:12200/vta/v1/jobs?page=1&size=5&sort=id
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
X-PageInfo-Total-Pages: 2
X-PageInfo-Size: 5
X-PageInfo-Total-Elements: 10
X-PageInfo-Page-Number: 1
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked
Date: Wed, 07 Oct 2015 14:20:55 GMT
```

In the sample above:

- X-PageInfo-Total-Pages: tells us that there is a total of two pages
- X-PageInfo-Size: of 5 jobs on each page
- X-PageInfo-Total-Elements: for a total of 10 elements in the database
- X-PageInfo-Page-Number: and this is the result of page 1

Here are some sample calls.

1.4.2 Getting page 1 with a size of 5 elements ordered by field ID

```
GET http://host:12200/vta/v1/jobs/transcode?page=1&size=5&sort=id
```

1.4.3 Getting a list of jobs with default paging with completionTime in descending order

```
GET http://host:12200/vta/v1/jobs/transcode?sort=completionTime,
desc
```

1.4.4 Getting a list of jobs with default paging ordered by completionTime and status

```
GET http://host:12200/vta/v1/jobs/transcode?sort=completionTime,
status
```

2 Transcoding

2.1 Using a preset

A job definition can refer to a preset by name. The job definition needs only to define the *presetName* property with a valid preset name. Each property of the preset will be imported at job creation time only if the corresponding job property has been left blank. This allows the overriding of any preset value at the job level, if applicable.

Note:
HLS output is created in a specific folder. The reason is that the HLS format is composed of multiple files. Each output consists of a playlist file and TS segment files. It is therefore convenient to have each HLS output in its own specific folder to avoid mixing the resulting files of multiple content versions together.

```
{
  "input": {
    "url": "file:///bigbuckbunny.mp4"
  },
  "outputs": [
    {
      "target": {
        "url": "file:///bigbuckbunny-HD/bigbuckbunny.m3u8"
      },
      "presetName" : "HD"
    }
  ]
}
```

2.2 Using a fully qualified job definition: No presets

The job definition model can refer to a preset or can fully qualify all the output properties for a given job. Most of the time presets are convenient, but there will be cases where the output definition is so specific that it's more convenient to use the fully qualified mode.

The following case shows how every output property and track can be defined in the job object instead of referring to a preset. This job will produce exactly the same result as the previous recipe.

```
{
  "input": {
    "url": "file:///bigbuckbunny.mp4"
  },
  "outputs": [
    {
      "target": {
        "url": "file:///bigbuckbunny-HD/bigbuckbunny.m3u8"
      },
      "format": {
        "type": "hls"
      },
      "tracks": [
        {
          "video":
            {
              "codecType": "h264",
              "qualityLevel": 5,
              "width": 1920,
              "height": 1080,
              "kfs": 2000,
              "framerate": "30000 / 1001",
              "advancedParams" : "profile=high",
              "bitrate": 8500000
            },
          "audio": [
            {
              "samplingRate": 48000,
              "channelMode" : "stereo",
              "codecType": "aac",
              "bitrate": 128000
            }
          ]
        }
      ]
    }
  ]
}
```

2.3 Customizing the transcoding properties

The job/preset definition allows customization of the output properties for a transcoding task. The properties are defined in the *tracks* array, where the *audio* and *video* track objects are located. They dictate the transcoding process. You can change everything: the codec, the resolution, the quality level (faster outputs versus better quality), the key frames, the frame rate, the bitrate and many more properties. An *advancedParams* property is also available at different levels to allow for providing non-conventional or encoder-specific properties.

Please refer to the *VTA API Specification* for more information about valid properties and values.

The following case shows various properties defined the *video* and *audio* objects.

```
{
  ...
  "outputs": [
    {
      ...
      "tracks": [
        {
          "video":
            {
              "codecType": "h264",
              "qualityLevel": 5,
              "width": 1920,
              "height": 1080,
              "kfs": 2000,
              "framerate": "30000 / 1001",
              "bitrate": 8500000
            },
          "audio": [
            {
              "samplingRate": 48000,
              "channelMode" : "stereo",
              "codecType": "aac",
              "bitrate": 128000
            }
          ]
        }
      ]
    }
  ]
}
```

2.4 Producing an HLS output

The VTA service can produce HLS outputs composed of variant playlists, playlists and media segments. As explained above, it is convenient to create the output files in a specific folder per content, which can be set in the target URL. When creating an HLS output, you can also choose to alter the default values of the container such as number of seconds per segment, the segment duration precision, the player's start offset and a few others.

The following is a sample HLS job with a single rate and segments of 15 seconds each. Below, you will also notice the *segmentDuration* property defined in the *format* (in milliseconds) to indicate 15-second segments.

```
 {
  "input":{
    "url": "file:///bigbuckbunny.mp4"
  },
  "outputs":[
    {
      "format":{
        "type":"hls",
        "segmentDuration": 15000
      },
      "target":{
        "url": "file:///bigbuckbunny-HD/bigbuckbunny.m3u8"
      },
      "tracks":[
        {
          "video":{
            "codecType":"h264",
            "width":320,
            "height":240,
            "bitrate":240000
          },
          "audio":[
            {
              "codecType":"aac",
              "bitrate":128000
            }
          ]
        }
      ]
    }
  ]
}
```

2.5 Producing an MP4 output

The format type to produce an MP4 file is *file.* That's the only main change from the previous example. Extra HLS oriented properties such as *segmentDuration* would be ignored if provided.

```
 {
  ...
  "outputs":[
    {
      "format":{
        "type":"file"
      },
      ...
    }
  ]
  ...
}
```

2.6 Multi-rate HLS content

HLS content can be generated with multiple tracks for streaming with adaptive bitrate. The VTA will produce a master playlist proposing the rates. Each rate will be in a subfolder with its own playlist and media segments.

See the folder arrangement below. The first level contains the master playlist and one folder per rate containing the segments and rate-specific playlist. The folder name represents a rate iterator and the video bitrate for easy referencing.

Multi-rate HLS content structure

```
.
| 1_1200000
|   |-- 57166a51e4b0acb13a6ca4b1-1_1200000_0.ts
|   |-- 57166a51e4b0acb13a6ca4b1-1_1200000_1.ts
|   |-- 57166a51e4b0acb13a6ca4b1-1_1200000_2.ts
|   `-- 57166a51e4b0acb13a6ca4b1-HLS-1_1200000.m3u8
|-- 2_1000000
|   |-- 57166a51e4b0acb13a6ca4b1-2_1000000_0.ts
|   |-- 57166a51e4b0acb13a6ca4b1-2_1000000_1.ts
|   |-- 57166a51e4b0acb13a6ca4b1-2_1000000_2.ts
|   `-- 57166a51e4b0acb13a6ca4b1-HLS-2_1000000.m3u8
`-- MASTER.m3u8
```

And here's the content of the MASTER master playlist:

```
MASTER playlist

#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1527600
1_1200000/57166a51e4b0acb13a6ca4b1-HLS-1_1200000.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=1297600
2_1000000/57166a51e4b0acb13a6ca4b1-HLS-2_1000000.m3u8
```

2.6.1 The Job object

The *job.outputs* object contains an array of audio/video *tracks* objects. The additional tracks are added in the *tracks* array. Each audio/video *tracks* object will create its own subfolder with playlist and segments. There is nothing else to do.

Here's a sample Job object:

```
Mutli-rate HLS job definition

{
   "input":{
      "url":"file:///var/cloud-ui/unlimited/sources/VIDEO0013.mp4"
   },
   "outputs":[
      {
         "format":{
            "type":"hls",
            "programId":1,
            "segmentDuration":10000,
            "segmentDurationPrecision":0,
            "segmentBufferSize":10
         },
         "target":{
            "url":"file:///VIDEO0013-20160419T172641/VIDEO0013.
m3u8"
         },
         "tracks":[
            {
               "video":{
                  "codecType":"h264",
                  "bitrate":1200000,
                  "width":640,
                  "height":360
               },
               "audio":[
                  {
                     "codecType":"aac",
                     "bitrate":96000
                  }
               ]
            },
            {
               "video":{
                  "codecType":"h264",
                  "bitrate":1000000,
                  "width":720,
                  "height":480
               },
               "audio":[
                  {
                     "codecType":"aac",
                     "bitrate":128000
                  }
               ]
            }
         ]
      }
   ]
}
```

2.7 Multi-variant HLS content

Offering a master playlist with multiple rates may not be enough for certain use cases. You might want to offer a master playlist with a different set of rates depending on the client. For instance, assume we offer both SD and HD rates for a certain content. If the client is low end, we may offer only the SD rate because HD is irrelevant for the client. In such a case, we use the *multiple variants* feature of the transcoding service.

Here's how such an example would look in terms of job definition:

Multiple HLS variants

```
{
   "input":{
      "url":"file:///var/cloud-ui/unlimited/sources/VIDEO0013.mp4"
   },
   "outputs":[
      {
         "format":{
            "type":"hls"
         },
         "target":{
            "url":"file:///C1/HD-SD.m3u8"
         },
         "tracks":[
            {
               "video":{
                  "codecType":"h264",
                  "bitrate":8500000,
                  "width":1920,
                  "height":1080
               },
               "audio":[
                  {
                     "codecType":"aac",
                     "bitrate":128000
                  }
               ]
            },
            {
               "video":{
                  "codecType":"h264",
                  "bitrate":3500000,
                  "width":960,
                  "height":540
               },
               "audio":[
                  {
                     "codecType":"aac",
                     "bitrate":128000
                  }
               ]
            }
         ]
      },
      {
         "format":{
            "type":"hls"
         },
         "target":{
            "url":"file:///C1/ONE-RATE.m3u8"
         },
         "tracks":[
            {
               "video":{
                  "codecType":"h264",
                  "bitrate":3500000,
                  "width":960,
                  "height":540
               },
               "audio":[
                  {
                     "codecType":"aac",
                     "bitrate":128000
                  }
               ]
            }
         ]
      }
   ]
}
```

Notice the target URLs. To be combined in a variant definition of the same content, each variant must have the same path and a specific m3u8 playlist name. Note the *tracks* objects as well. They are copied into each output with the same set of fields and will be bound to the same transcoded track object.

This job would produce the following content and folder structure:

HLS variants content structure

```
.
|-- 1_8500000
|   |-- 5716866fe4b028076f90fe64-1_8500000_0.ts
|   |-- 5716866fe4b028076f90fe64-1_8500000_1.ts
|   |-- 5716866fe4b028076f90fe64-1_8500000_2.ts
|   `-- 5716866fe4b028076f90fe64-HLS-1_8500000.m3u8
|-- 2_3500000
|   |-- 5716866fe4b028076f90fe64-2_3500000_0.ts
|   |-- 5716866fe4b028076f90fe64-2_3500000_1.ts
|   |-- 5716866fe4b028076f90fe64-2_3500000_2.ts
|   `-- 5716866fe4b028076f90fe64-HLS-2_3500000.m3u8
|-- HD-SD.m3u8
`-- SD.m3u8
```

And here's the content of the HD-SD master playlist:

HD-SD playlist

```
#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=9057600
1_8500000/5716866fe4b028076f90fe64-HLS-1_8500000.m3u8
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=3927600
```

```
2_3500000/5716866fe4b028076f90fe64-HLS-2_3500000.m3u8
```

And, finally, the SD-only master playlist:

```
SD playlist

#EXTM3U
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=3927600
2_3500000/5716866fe4b028076f90fe64-HLS-2_3500000.m3u8
```

2.8 MPEG-TS

VTA can also read a single source and produce *mpegts* outputs for live streaming.

The solution supports the following two models:

1. Multiple single-rate SPTS (**S**ingle **P**rogram **T**ransport **S**tream)
2. Single multi-rate SPTS

Multiple single-rate SPTS

In this case, the user wants a single rate per output stream. He creates one output per stream and specifies the target *url* and the *track* definition having a single video and audio. Note that the *format.type* must be the same in each output.

```
Multiple single-rate SPTS

{
  "input": {
    "url": "udp://172.16.202.20:28000",
    "type" : "live"
  },
  "outputs": [
    {
      "format": {
        "type": "mpegts"
      },
      "target": {
        "url": "udp://232.88.0.112:8961",
        "datarate": 6500000
      },
      "tracks": [
        {
            "programId": 153,
            "video":
              {
                  "codecType": "h264",
                  "width": 352,
                  "height": 240,
                  "bitrate": 750000
              },
            "audio": [
              {
                  "codecType": "aac",
                  "bitrate": 128000
              }
            ]
          }
        ]
      },
      {
        "format": {
          "type": "mpegts"
        },
        "target": {
          "url": "udp://232.88.0.112:8963",
          "datarate": 7500000
        },
        "tracks": [
          {
            "programId": 154,
            "video":
              {
                  "codecType": "h264",
                  "width": 640,
                  "height": 480,
                  "bitrate": 1500000
              },
            "audio": [
              {
                  "codecType": "aac",
                  "bitrate": 128000
              }
            ]
          }
        ]
      }
    ]
  }
```

Single multi-rate SPTS

The next case has a single output with a single video track but multiple audio tracks all in the same program.

Single multi-rate SPTS

```
{
  "input": {
    "url": "udp://172.16.202.20:28000",
    "type" : "live"
  },
  "outputs": [
    {
      "format": {
        "type": "mpegts"
      },
      "target": {
        "url": "udp://232.88.0.112:8961",
        "datarate": 6500000,
        "advancedParams": "pcrDelta=12,someOtherParam=123"
      },
      "tracks": [
        {
          "programId": 153,
          "advancedParams": "pctPid=0x00",
          "video":
            {
                "codecType": "h264",
                "width": 352,
                "height": 240,
                "bitrate": 750000
            },
          "audio": [
            {
                "codecType": "aac",
                "bitrate": 128000
            },
            {
                "codecType": "aac",
                "bitrate": 64000
            }
          ]
        }
      ]
    }
  ]
}
```

Notes:

- *If the source has many streams, the job definition must specify which stream to use; otherwise, the decoder will pick the first one.*
- *There is a third case supported by the API (multi-program per output) but not yet supported at the encoding level so we will skip this discussion.*

2.9 Subtitles and Closed Captions

The VTA supports carrying subtitles and closed captions from a source to a target file during a transcoding operation. Conceptually, an output defines one or more tracks. Each track may have audio and/or video elements. The subtitles and closed captions support comes with a “Text []” element array in the track. The track can have one or more text elements, depending on the source constituents.

The VTA service supports EIA-608 and EIA-708 for both input and output. The 608 versus 708 is determined by the *sourceChannel* structure, which is CC[1-4] for the EIA-608 and SERVICE [01-64] for the EIA-708 standard. The VTA service can also output WebVTT format. This is done using the *outOfBand* versus *inBand* text type.

It is important to note that whether the closed captions need to be declared in order to be included in the output depends on the transcoder release. Newer versions allow for automatic passthrough without further specification.

Here's an example of a Text declaration:

```
"texts": [
  {
    "type": "outOfBand",
    "sourceChannel": "CC1",
    "hlsProperties": {
        "name": "English",
        "language": "en-US",
        "default": true,
        "autoselect": false,
        "forced": false
    }
  }
]
```

In this example, the output would contain an *outOfBand* set of captions (WebVTT) having the captions of the “CC1” source channel. The player will display these captions as “English” and will play this text in the absence of information from the user indicating a different choice .

2.9.1 Closed-captions use cases

Here are a few typical supported use cases:

- In-band single- or multi-text
- TS input to MP4 output (limited support)
- TS input to TS output
- TS input to HLS output
- Out-of-band single- or multi-text

2.9.2 Sample #1: Single in-band channel

This is the simplest case: one in-band text channel over a non-HLS format. The only information required is the text type (inBand) and the channel source.

```
{
  ...
  "outputs":[
    {
      ...
      "tracks":[
        {
          ...
          "texts":[
            {
              "type":"inBand",
              "sourceChannel":"CC1"
            }
          ]
        }
      ]
    }
  ]
}
```

2.9.3 Sample #2: Single in-band channel (HLS)

In this sample, we use an "hls" format and add the "hlsProperties" in the "text" object in order to produce the additional tags required in for HLS playlists.

```
{
  ...
  "outputs":[
    {
      ...
      "tracks":[
        {
          ...
          "texts":[
            {
              "type":"inBand",
              "sourceChannel":"CC1",
              "hlsProperties": {
                "name": "English",
                   "language": "en",
                   "default": true,
                   "forced": true,
                   "autoselect": true
               }
            }
          ]
        }
      ]
    }
  ]
}
```

2.9.4 Sample #3: Single out-of-band channel (HLS)

This sample is very similar to Sample #2 but we use an "outOfBand" text type generating subtitles and WebVTT instead of closed captions.

```
{
  ...
  "outputs":[
    {
      ...
      "tracks":[
        {
          ...
          "texts":[
            {
              "type":"outOfBand",
              "sourceChannel":"CC1",
              "hlsProperties": {
                "name": "English",
                   "language": "en",
                   "default": true,
                   "forced": true,
                   "autoselect": true
               }
            }
          ]
        }
      ]
    }
  ]
}
```

2.9.5 Sample #4: Multi-text out-of-band channel (HLS)

We add a second "French" text channel. This will generate a second subtrack in outputs and add a variant to the group.

```
{
  ...
  "outputs":[
    {
      ...
      "tracks":[
        {
          ...
          "texts":[
            {
              "type":"outOfBand",
              "sourceChannel":"CC1",
              "hlsProperties": {
                "name": "English",
                   "language": "en",
                   "default": true,
                   "forced": true,
                   "autoselect": true
               }
            },
            {
              "type":"outOfBand",
              "sourceChannel":"CC2",
              "hlsProperties": {
                "name": "French",
                   "language": "fr",
                   "default": false,
                   "forced": true,
                   "autoselect": true
               }
            }
          ]
        }
      ]
    }
  ]
}
```

2.9.6 Sample #5: Multi-track/Multi-text out-of-band channels (HLS)

Here we add another track with a different set of channels. The first track has two channels (English and French) and second one has only English. This will create a second subtitles group and assign the proper group to each rate.

```
{
  ...
  "outputs":[
    {
      ...
      "tracks":[
        {
          ...
          "texts":[
            {
              "type":"outOfBand",
              "sourceChannel":"CC1",
              "hlsProperties": {
                "name": "English",
                   "language": "en",
                   "default": true,
                   "forced": true,
                   "autoselect": true
               }
            },
            {
              "type":"outOfBand",
              "sourceChannel":"CC2",
              "hlsProperties": {
                "name": "French",
                   "language": "fr",
                   "default": false,
                   "forced": true,
                   "autoselect": true
               }
            }
          ]
        },
        {
          ...
          "texts":[
            {
              "type":"outOfBand",
              "sourceChannel":"CC1",
              "hlsProperties": {
                "name": "English",
                   "language": "en",
                   "default": true,
                   "forced": true,
                   "autoselect": true
               }
            }
          ]
        }
      ]
    }
  ]
}
```

2.9.7 Sample #6: Multi-variant/Multi-track/Multi-text out-of-band channels (HLS)

Here is the most complex of our samples. It declares a second HLS variant having a single rate with a Spanish channel being CC3. This will create a third group for Spanish and a third rate with a reference to this group.

```
{
  ...
  "outputs":[
    {
      ...
      "tracks":[
        {
            "texts":[
              {
                "type":"outOfBand",
                "sourceChannel":"CC1",
                "hlsProperties": {
                  "name": "English",
                     "language": "en",
                     "default": true,
                     "forced": true,
                     "autoselect": true
                 }
              },
              {
                "type":"outOfBand",
                "sourceChannel":"CC2",
                "hlsProperties": {
                  "name": "French",
                     "language": "fr",
                     "default": false,
                     "forced": true,
                     "autoselect": true
                 }
              }

            ]
          },
          {
            ...
            "texts":[
              {
                "type":"outOfBand",
                "sourceChannel":"CC1",
                "hlsProperties": {
                  "name": "English",
                     "language": "en",
                     "default": true,
                     "forced": true,
                     "autoselect": true
                 }
              }
            ]
          }
        ]
      },
      {
        ...
        "tracks":[
          {
            ...
            "texts":[
              {
                "type":"outOfBand",
                "sourceChannel":"CC3",
                "hlsProperties": {
                  "name": "Spanish",
                     "language": "sp",
                     "default": false,
                     "forced": true,
                     "autoselect": true
                 }
              }
            ]
          }
        ]
      }
    ]
  }
```

2.10 Using plugins

The input and output objects can contain plugin definitions. The plugin may refer to native operations or a third-party library. Typically, a plugin will apply an operation on the stream. Apart from a few exceptions, the plugin can be set in the input or output object.

The most commonly used plugin types are:

- Deinterlacing
- Overlays (image, text, bitrate, and so on)
- Dewarping

Here's a sample deinterlacing (deinterlacing can only be set on the input stream).

```
{
  "input": {
    ...
    "video": {
      "plugins": [
        {
          "type" : "deinterlacer",
          "name" : "interdigital",
          "params": {
             "prop1" : "value1",
             "prop2" : "value2"
          }
        }
      ]
    }
  },
  "outputs": [
   ...
  ]
}
```

The following example shows a dewarper sample with overlays.

```
{
  "input": {
    ...
    "video": {
      "plugins": [
        {
          "type": "dewarper",
          "name": "Dewarper_Cam",
          "params": {
            "position" : "ceiling",
            "lensType" : "A0IFV",
            "width" : "640",
            "height" : "480",
            "pan" : "15.0",
            "tilt" : "80.0",
            "zoom" : "81.0",
            "viewingMode" : "single"
          }
        }
      ]
    }
  },
  "outputs": [
   {
      ...
      "tracks": [
      {
        "video": {
          ...
          "plugins": [
           {
              "type": "textOverlay",
              "name": "hello world",
              "params": {
                "text" : "Hello World"
               }
           },
           {
              "type": "clockOverlay",
              "name": "clockBlack",
              "params": {
                "red" : "200",
                "green" : "200",
                "blue" : "200"
              }
            }
          ]
        }
      }
      ],
      ...
    }
  ]
}
```

2.11 Using notifications

The VTA service provides a notification mechanism allowing a client to be notified on specific events. The current version supports a single notification message: the job is completed. When a job object is posted, a notification object can be included. This will instruct the VTA service to send a message to the specified sink on job completion.

The job object:

```
{
 ...
 "notification": {
    "url": "http://sink-host/vta"
  }
}
```

The notification object contains a sink URL and a POST method. The VTA currently supports only HTTP POST notification. The VTA will post the following object structure:

```
{
    id: String
    status: JobStatus
    input: Input
    outputs: Output[]
}
```

2.12 DVB subtitles and teletext

We saw earlier a few examples of Closed Captions and Subtitles support. The VTA abstracts these notions in a *Text* element added in the tracks definition. Text elements can be combined with audio and video to form a complete track of audio, video and one or more text tracks that a user can choose from on his device. The EIA-608 and EIA-708 Closed Captions are referenced in the source media using a channel identifier (CC1-CC4, SERVICE01-64). DVB subtitles and teletext are slightly different. The reference to the source track is done using a *language* or a *pid* . The text elements are packaged as a separate track like for audio and video and are treated as such by the encoder. If no input track selector information is provided, the encoder will pick up the first one it sees (that is included in the selected program for MPEG-TS inputs).

In order to better understand this, here's a content characterization containing dvbsub and teletext elements:

```
Content with dvbsub & teletext

AUDIO                            CodecName audio/mpeg
AUDIO                              Version 1
AUDIO                             Duration 215.5200
AUDIO                               TsChan 636
AUDIO                                TsPid 0x1DD7
AUDIO                           StreamSize 457870
AUDIO                              TrackId 7639
AUDIO                       AverageBitrate 191.9790
AUDIO                             Language eng

VIDEO                            CodecName video/mpeg
VIDEO                              Version 2
VIDEO                             Duration 215.6000
VIDEO                               TsChan 636
VIDEO                                TsPid 0x1CA8
VIDEO                           StreamSize 6858416
VIDEO                              TrackId 7336
VIDEO                                Width 720
VIDEO                               Height 576
VIDEO                       AverageBitrate 2918.4749

TEXT                             CodecName private/dvbsub
TEXT                              Duration 6.6933
TEXT                                TsChan 636
TEXT                                 TsPid 0x1F66
TEXT                             StreamSize 6731
TEXT                                TrackId 8038
TEXT                         AverageBitrate 11.9817
TEXT                               Language lit

AUDIO_0001                        CodecName audio/mpeg
AUDIO_0001                          Version 1
AUDIO_0001                         Duration 215.5200
AUDIO_0001                           TsChan 636
AUDIO_0001                            TsPid 0x1DD8
AUDIO_0001                       StreamSize 381734
AUDIO_0001                          TrackId 7640
AUDIO_0001                   AverageBitrate 160.0562
AUDIO_0001                         Language rus

TEXT_0001                         CodecName private/teletext
TEXT_0001                          Duration 215.5400
TEXT_0001                            TsChan 636
TEXT_0001                             TsPid 0x1F69
TEXT_0001                        StreamSize 513606
TEXT_0001                           TrackId 8041
TEXT_0001                    AverageBitrate 215.1229
TEXT_0001                          Language lit
```

In this content, we can see two text tracks: a *dvbsub* on track id 8038 and *teletext* on track id 8041.

To define the transcoding job to copy the text elements to the target, you need to add a *Text* element using the type *dvbsub* or *teletext* according to the type of the text track as well as input track selection information since there is more than one track in the program.

Here's a sample based on this content:

Job definition for dvbsub & teletext

```
{
  "input":{
    "url":"file:///opt/sample-medias/bigbuckbunny.mp4",
    "type":"vod"
  },
  "outputs":[
    {
      "target":{
        "url":"file:/hls/bigbuckbunny"
      },
      "format":{
        "type":"hls"
      },
        "tracks":[
          {
            "audio":[
              {
                "codecType":"aac",
                "bitrate":128000
              }
            ],
            "video":{
              "codecType":"h264",
              "bitrate":3500000,
              "width":1280,
              "height":720
            },
            "texts":[
              {
                "type":"dvbsub",
                "inputTrackSelector": "pid=8038"
              },
              {
                "type":"teletext",
                "inputTrackSelector": "pid=8041"
              }
            ]
          }
        ]
      }
    ]
  }
```

2.13 Multi-language audio transcoding

In this section, we look into how the VTA supports the transcoding and HLS packaging of content that has multiple audio tracks. A content can have multiple audio tracks to provide different levels of quality, different languages or both. The VTA allows the selection of the audio track to add in the target as well as the definition of the corresponding transcoding properties. The input audio track selection is similar to text track selection. It can be done using the language code, if available, or the pid.

Alternate renditions

The HLS playlist format supports alternate renditions for a given audio or video track. For instance, this feature allows support of multiple languages for audio. It is off by default in VTA and needs to be turned on using the *job.output.format.alternateRenditions* property. This will instruct the encoder to create the tags necessary to define and indicate the alternate audio tracks. The *default* property will tell which audio is played by default without any additional player instructions.

Multiple tracks

You will notice in the example below that the video and audio tracks are all separately packaged as tracks in the output. The video is encoded separately without audio and each audio is separate, without any video. This is required in order to allow dynamic audio switching.

Multi-Language Audio Transcoding

```
{
   "input":{
      "url":"file:///opt/sample-medias/multiaudio.ts"
   },
   "outputs":[
      {
         "format":{
            "type":"hls",
            "alternateRenditions": true
         },
         "target":{
            "url":"file:/ml-single/eng-fre-ger"
         },
         "tracks":[
            {
               "name":"video",
               "video":{
                  "codecType":"h264",
                  "width":320,
                  "height":240,
                  "bitrate":200000
               }
            },
            {
               "name":"audio_english",
               "audio":[
                  {
                     "inputTrackSelector": "language=eng",
                     "codecType":"aac",
                     "bitrate":64000,
                     "default":true,
                     "autoselect": true
                  }
               ]
            },
            {
               "name":"audio_french",
               "audio":[
                  {
                     "inputTrackSelector": "language=fra",
                     "codecType":"aac",
                     "bitrate":64000
                  }
               ]
            },
            {
               "name":"audio_german",
               "audio":[
                  {
                     "inputTrackSelector": "language=ger",
                     "codecType":"aac",
                     "bitrate":64000
                  }
               ]
            }
         ]
      }
   ]
}
```

The content generated will be structured with four different tracks, each having its playlist, with a master playlist on top of them all.

```
Content files

.
`-- ml-single
    |-- audio_english
    |   |-- 5745e9e2e4b014811abc8956-audio_english_0.ts
    |   |-- 5745e9e2e4b014811abc8956-audio_english_1.ts
    |   |-- 5745e9e2e4b014811abc8956-audio_english_2.ts
    |   `-- 5745e9e2e4b014811abc8956-HLS-audio_english.m3u8
    |-- audio_french
    |   |-- 5745e9e2e4b014811abc8956-audio_french_0.ts
    |   |-- 5745e9e2e4b014811abc8956-audio_french_1.ts
    |   |-- 5745e9e2e4b014811abc8956-audio_french_2.ts
    |   `-- 5745e9e2e4b014811abc8956-HLS-audio_french.m3u8
    |-- audio_german
    |   |-- 5745e9e2e4b014811abc8956-audio_german_0.ts
    |   |-- 5745e9e2e4b014811abc8956-audio_german_1.ts
    |   |-- 5745e9e2e4b014811abc8956-audio_german_2.ts
    |   `-- 5745e9e2e4b014811abc8956-HLS-audio_german.m3u8
    |-- eng-fre-ger.m3u8
    `-- video
        |-- 5745e9e2e4b014811abc8956-HLS-video.m3u8
        |-- 5745e9e2e4b014811abc8956-video_0.ts
        |-- 5745e9e2e4b014811abc8956-video_1.ts
        `-- 5745e9e2e4b014811abc8956-video_2.ts
```

And the master playlist will read:

Master playlist

```
#EXTM3U
#EXT-X-VERSION:4
#EXT-X-MEDIA:URI="audio_english/5745e9e2e4b014811abc8956-HLS-
audio_english.m3u8",TYPE=AUDIO,GROUP-ID="Group-1",LANGUAGE="eng",
NAME="Variant-1",DEFAULT=YES,AUTOSELECT=YES,FORCED=NO
#EXT-X-MEDIA:URI="audio_french/5745e9e2e4b014811abc8956-HLS-
audio_french.m3u8",TYPE=AUDIO,GROUP-ID="Group-1",LANGUAGE="fra",
NAME="Variant-2",DEFAULT=NO,AUTOSELECT=NO,FORCED=NO
#EXT-X-MEDIA:URI="audio_german/5745e9e2e4b014811abc8956-HLS-
audio_german.m3u8",TYPE=AUDIO,GROUP-ID="Group-1",LANGUAGE="ger",
NAME="Variant-3",DEFAULT=NO,AUTOSELECT=NO,FORCED=NO
#EXT-X-STREAM-INF:PROGRAM-ID=1,BANDWIDTH=280000,AUDIO="Group-1"
video/5745e9e2e4b014811abc8956-HLS-video.m3u8
```

2.14 Passthrough codec

Sometimes we only want to transcode either the video or audio in a content and let the other pass through untouched. The VTA supports a special *passthrough* codec for this purpose. Using this codec will tell the encoder to copy the track as it is in the source without any decoding /encoding.

The passthrough codec is supported for both audio and video with the same syntax.

See the sample below, where the audio is passed through the encoder without any transcoding:

Passthrough codec

```
{
  ...
  "outputs": [
    {
        ...
        "tracks": [
          {
            "video":
              {
                "codecType": "h264",
                "qualityLevel": 5,
                "width": 1920,
                "height": 1080,
                "kfs": 2000,
                "framerate": "30000 / 1001",
                "advancedParams" : "profile=high",
                "bitrate": 8500000
              },
            "audio": [
              {
                "codecType": "passthrough"
              }
            ]
          }
        ]
      }
    ]
  }
```

2.15 Adding Blackouts

A channel can be created with blackout periods. Blackouts can be triggered by a schedule using a schedule file (already pre-loaded on the server) or can be triggered using a signal in the blackout source. This is controlled using a *blackout mode*. There are 3 modes supported by VTA:

1. *scheduled*: The regular feed is used by default when no blackout schedule file exists
2. *scheduledDefaultOnBlackout*: The blackout feed is used by default when no blackout schedule file exists
3. *triggeredByBlackoutFeed:* The blackout feed is used to trigger the blackout state

The blackout information is added in the job at creation time using the compound property *blackout.* The blackout definition requires a replacement *input* definition as well as a schedule. The default mode is "*scheduled".* When using schedule-based modes (*scheduled & scheduledDefaultOnBlackout),* a *schedule* property must be added in order to indicate where to find the schedule file (*url)* and what is the timezone of the schedule file.The timezone default value is "UTC" and accepts standard timezone strings such as "America/New_York".

See an example below:

```
{
  "input": {
    "url": "udp://172.16.202.20:28000",
    "type" : "live"
  },
  "blackout": {
    "mode": "scheduled",
    "schedule": {
      "url": "file:///this/is/the/schedule/file"
    },
    "input": {
      "url": "udp://172.16.202.22:38000",
      "type" : "live"
    }
  },
  "outputs": [
    {
      "format": {
        "type": "mpegts"
      },
      "target": {
        "url": "udp://239.255.255.23:26177"
      },
      "presetName" : "SD"
    }
  ]
}
```