Task-Engine Documentation

VUALTO

Jul 13, 2023

Contents

The Task Engine is a product that facilitates the creation of VOD from online and offline sources. It is designed to allow simple or complicated custom workflows to be developed, deployed and maintained easily and quickly. The Task Engine is exposed via a REST API and reports back to the client system via a callback mechanism.

CHAPTER 1

How it works

The Task Engine breaks a workflow up in to several component parts. Every piece of work submitted is a job, and each job is broken up in to a series of tasks. These tasks are then scheduled on to queues which workers then process.

Every job and task have unique identities, and these are exposed back to the client system both through the response to the initial job submission and through the callbacks (unless for some reason the configuration prevents this from happening).

Callbacks are sent to the specified endpoints when each task starts and ends (successful or fail). This allows the integrated systems to keep track of the job's progress. A final callback is also submitted when a job completes. The final callback will always contain the status of the job (success or fail) and information about the assets related to the job.

CHAPTER 2

Contents:

2.1 DEVELOPER DOCUMENTATION

As mentioned in the index page the Task Engine facilitates the creation of VOD from online and offline sources. This supports products such as VCH, Clip2VU and Media Syndication.

2.1.1 INTEGRATION

There are two main integration points for the Task Engine:

API – how jobs are submitted to the Task Engine.

Callbacks – how the Task Engine notifies client systems of job progress.

API

The API is mainly used to trigger workflows within the Task Engine but additional API endpoints are available for job management. Full API documentation can be found here

CALLBACKS

Callbacks are used to notify a integrated services with workflow execution updates. Callback URLs are submitted as part of the payloads and the Task Engine will send callbacks when:

1. A task starts
2. A task ends (success or fail)
3. A job ends (success or fail)

All default task callbacks will the contain the same JSON body structure:

```
"job_id": "<job id>",
"task_id": "<task id>",
"task_name": "<task name>",
"workflow": "<workflow name>",
"event": "<task event>",
"content_id": "<content_id>",
"message": "<task exception message>"
```

Job callbacks vary depending on the workflow being executed but the following are common in all workflows.

```
"job_id": "<job id>",
"status": "<job status>",
"workflow": "<workflow name>",
"content_id": "<content id>",
"custom_data": "<client custom data>"
```

More information of the callbacks for each workflow can be found here.

Specifying the Vualto Control Hub Video Information Service web-hook (`https://vis.controlhub.[client].vualto.com/api/event/vuflow/taskenginecallback`) as a callback url, will be add the asset as a VOD event within the Vualto Control Hub CMS. A second CMS web-hook (`https://admin.controlhub.[client].vualto.com/vod/PublishVuflowData`) can also be included for realtime updates on the status of the job.

As Task Engine is an integration product that can be customised, any specific requirements are easily catered for (eg. setting authentication headers) and the body of a callbacks may also be modified.

AUTHENTICATION

All Task Engine API calls that require authentication currently use the client name and API key provided by Vualto. The credentials should be supplied as `client` and `api-key` headers respectively.

2.1.2 TASK ENGINE API

The Task Engine endpoints will always return a JSON response unless explicitly indicated otherwise.

STATUS ENDPOINTS

GET: /

This endpoint will check if the Task Engine endpoint is reachable.

Requires Authentication: No Required Headers: None Optional Headers: None

```
{
    "result": "alive"
}
```

GET: `/health`

The health endpoint will run checks on the different Task Engine components and returns the status of each service. The endpoint will also return some information about the Task Engine and statistics about jobs and tasks.

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication

Optional Headers: None

Successful Response:

```
{
    "version": "1.169.3", // Task Engine version
    "databases": {
        "redis": "OK",
        "postgres": "OK"
    },
    "queues": {
        "windows_capture": {
            "workers": 12,
            "working": 0,
            "pending": 0
        },
        "scheduler": {
            "workers": 1,
            "working": 0,
            "pending": 0
        },
        "work": {
            "workers": 9,
            "working": 0,
            "pending": 0
        },
        "controller": {
            "workers": 1,
            "working": 0,
            "pending": 0
        },
        "callback": {
            "workers": 2,
            "working": 0,
            "pending": 0
        }
    },
    "tasks": {
        "failed": 0,
        "pending": 0,
        "processed": 987654321
    },
    "jobs": {
        "completed": 12345,
        "failed": 65,
        "pending": 0,
        "broken": 20,
        "queued": 4,
        "started": 8,
        "scheduled": 7,
        "paused": 0,
        "max_jobs": 8,
        "priority_slots": 3
```

(continues on next page)

(continued from previous page)

```
    }
}
```

500 - Error Response:

```
{
    "error": "<error message>"
}
```

GET: `/dashboard`

The dashboard endpoint returns information about the current Task Engine queue status. The information includes lists of started, queued and scheduled jobs as well as the setting information for the maximum concurrent jobs and the number of priority reserved job slots. The Task Engine version is also returned.

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication

Optional Headers: None

Successful Response:

```
{
    "max_jobs": 2,
    "priority_slots": 0,
    "started": [
        {
            "id": 123,
            "client": "demo-client",
            "workflow": "vodcapture",
            "priority": 5,
            "position": 1,
            "created_at": "2019-11-29T10:47:50.431Z",
            "updated_at": "2020-07-13T09:44:48.451Z",
            "queue_state": "started",
            "failed": false,
            "run_at": "2020-11-29T18:00:30.000Z"
        },
        {
            "id": 124,
            "client": "demo-client",
            "workflow": "vodcapture",
            "priority": 5,
            "position": 1,
            "created_at": "2020-06-19T15:23:36.197Z",
            "updated_at": "2020-09-07T15:28:12.034Z",
            "queue_state": "started",
            "failed": false,
            "run_at": "2030-06-19T17:00:30.000Z"
        }
    ],
    "queued": [
        {
```

(continues on next page)

(continued from previous page)

```
            "id": 125,
            "client": "demo-client",
            "workflow": "vodcapture",
            "priority": 6,
            "position": 1,
            "created_at": "2019-11-29T10:47:50.431Z",
            "updated_at": "2020-07-13T09:44:48.451Z",
            "queue_state": "queued",
            "failed": false,
            "run_at": "2020-11-29T18:00:30.000Z"
        },
        {
            "id": 126,
            "client": "demo-client",
            "workflow": "vodcapture",
            "priority": 6,
            "position": 1,
            "created_at": "2020-06-19T15:23:36.197Z",
            "updated_at": "2020-09-07T15:28:12.034Z",
            "queue_state": "queued",
            "failed": false,
            "run_at": "2030-06-19T17:00:30.000Z"
        }
    ],
    "scheduled": [
        {
            "id": 122,
            "client": "demo-client",
            "workflow": "vodcapture",
            "priority": 5,
            "position": 1,
            "created_at": "2020-07-13T09:49:27.086Z",
            "updated_at": "2020-07-13T09:49:27.206Z",
            "queue_state": "scheduled",
            "failed": false,
            "run_at": "2030-06-19T17:00:30.000Z"
        }
    ],
    "version": "1.169.3"
}
```

JOB ENDPOINTS

POST: `/job`

This endpoint is used to submit jobs to the Task Engine. It is the endpoint used most often. The payload for this endpoint varies substantially depending on the workflow being submitted. More information on the payload properties for each workflow can be found here. Successful job submission will return an `accepted` result and the job id. An error message is returned when a job submission fails.

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication

- `Content-Type` - set to `application/json`

Optional Headers: None

Successful Response:

~~~
{
    "id": "<job id>",
    "result": "accepted"
}


400 - Error Response:

```json
{
    "id": "<job id>",
    "error": "<error message>"
}
~~~

500 - Error Response:

```
    "Unable to create job request"
```

GET: `/jobs`

This endpoint is used to return a list of jobs from the Task Engine database. Filtering is supported through query string parameters. The default search (no parameters) will return the last 10 jobs.

Requires Authentication: No Required Headers: None Optional Headers:None Query String Parameters:

- `limit` - the maximum number of jobs to return
- `order_by` - order the query by a job property
- `asc` - order the results in ascending or descending order. Accepts 1 (true) or 0 (false)
- `state` - filter by job state. One of the following IDs needs to be specified
  - 0 - queued
  - 1 - started
  - 2 - completed
  - 3 - pending
  - 4 - broken
  - 5 - scheduled
  - 6 - paused
- `from` - used to filter by date range, based on the job creation date
- `to` - used to filter by date range, based on the job creation date
- `failed` - returned failed jobs. If used with a state, jobs will only be returned if state is set to 2 (completed)
- `search` - search term used to filter by eg. the content id for a submitted job
- `job_ids` - comma separated job ids
- `client` - client name to filter by

- `persist` - filter for jobs set to persist. Accepts 1 (true) or 0 (false)

Successful response for `/jobs?limit=3&client=demo-client&state=2`

```
[
    {
        "id": 123,
        "client": "demo-client",
        "workflow": "vodcapture",
        "priority": 5,
        "position": 1,
        "created_at": "2020-09-24T11:55:33.755Z",
        "updated_at": "2020-09-24T12:17:33.566Z",
        "queue_state": "completed",
        "parameters": "<parameters submitted when creating the job>",
        "failed": false,
        "run_at": "2020-09-24T11:55:33.755Z"
    },
    {
        "id": 114,
        "client": "demo-client",
        "workflow": "vodcapture",
        "priority": 5,
        "position": 1,
        "created_at": "2020-09-24T11:55:32.971Z",
        "updated_at": "2020-09-24T12:05:18.937Z",
        "queue_state": "completed",
        "parameters": "<parameters submitted when creating the job>",
        "failed": false,
        "run_at": "2020-09-24T11:55:32.971Z"
    },
    {
        "id": 102,
        "client": "demo-client",
        "workflow": "vodcapture",
        "priority": 5,
        "position": 1,
        "created_at": "2020-09-24T11:55:32.031Z",
        "updated_at": "2020-09-24T12:18:33.641Z",
        "queue_state": "completed",
        "parameters": "<parameters submitted when creating the job>",
        "failed": false,
        "run_at": "2020-09-24T11:55:32.031Z"
    }
]
```

400 - Error Response:

```
{
    "error": "<error message>"
}
```

GET: `/jobs/<job_id>`

This endpoints returns information about the specified job.

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication

Optional Headers: None

Successful response for `/jobs/123`

```
{
    "id": 123,
    "client": "demo-client",
    "workflow": "vodstream",
    "priority": 5,
    "position": 1,
    "top_of_queue": false,
    "parameters": "<parameters submitted when creating the job>",
    "created_at": "2019-09-18T17:13:47.713Z",
    "updated_at": "2019-09-18T17:16:05.215Z",
    "queue_state": "completed",
    "failed": false,
    "run_at": "2019-09-18T17:13:47.713Z"
}
```

400 - Error Response:

```
{
    "error": "<error message>"
}
```

PATCH: `/jobs/<job_id>`

This endpoint is used to update job fields. Only a specific selection of fields can be updated after a job has been submitted. The response will return the job id and the result of the update.

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication
- `Content-Type` - set to `application/json`

Optional Headers: None

The list of job fields that can be updated:

- `queue_state` - The queue state for a job can be updated. This can be used to pause or break a job by specifying the values `paused` or `broken` respectively
- `run_at` - Updating the run_at field for a job changes when the job will be queued. The date must be in UTC and in the following format `yyyy-MM-ddTHH:mm:ss.fff`
- `priority` - Updating the priority for a job. More information on job priority can be found here
- `sempahore_url` - This url can be used as part of the scheduling process. More information on the semaphore url can be found here
- `persist` - This boolean flag is used to indicate that the job and its logs should persist and not be cleared as part of any automatic database clean up

Sample payload:

```
{
    "client": "demo-client",
    "priority": "2",
    "run_at": "2020-09-09T14:30:00.000"
}
```

Successful Response:

```
{
    "id": "<job id>",
    "result": "Performed updates: <list of updates>"
}
```

400 - Error Response:

```
{
    "id": "<job id>",
    "error": "<error message>"
}
```

POST: `/jobs/<job id>/rerun`

This endpoint is used to rerun a job with exactly the same parameters. When rerunning a job, the original job's queue state will be set to broken since the content it generated will no longer be valid.

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication

Optional Headers: None

Successful Response:

```
{
    "id": "<job id>",
    "result": "accepted"
}
```

400 - Error Response:

```
{
    "error": "<error message>"
}
```

LOG ENDPOINTS

GET: `/logs/<job id>`

This endpoint is used to retrieve the logs for the specified job.

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication

- `Accept` - set to `application/json`

Optional Headers: None

Successful Response:

```
[
    {
        "id": 691464,
        "job_id": 123,
        "severity": 1,
        "severity_description": "INFO",
        "progname": "api",
        "message": "{\"path\":\"POST /job\",\"remote_addr\":\"99.80.104.160\",\
↪"headers\":{\"HTTP_VERSION\":\"HTTP/1.1\",\"HTTP_X_AUTH_KEY\":\
↪"********************************3e2d\",\"HTTP_API_KEY\":\
↪"********************************3e2d\",\"HTTP_X_API_KEY\":\
↪"********************************3e2d\",\"HTTP_ACCEPT\":\"application/json,␣
↪application/xml, text/json, text/x-json, text/javascript, text/xml\",\"HTTP_USER_
↪AGENT\":\"RestSharp/106.3.1.0\",\"HTTP_HOST\":\"taskengine.demo-client.vualto.com\",
↪\"HTTP_ACCEPT_ENCODING\":\"gzip, deflate\"},\"parameters\":{\"client\":\"demo-
↪client\",\"parameters\":{\"folder\":\"a40fdaff-f904-4ea5-b893-a89152708952\",\
↪"content_id\":\"a40fdaff-f904-4ea5-b893-a89152708952\",\"rest_endpoints\":[\"https:/
↪/vis.controlhub.demo-client.vualto.com/api/event/vuflow/taskenginecallback\",\
↪"https://admin.controlhub.demo-client.vualto.com/vod/PublishVuflowData\"]},\"job\":
↪{\"workflow\":\"drmswitch\"}},\"payload\":\"job_created\"}",
        "created_at": "2020-07-06T12:48:49.591Z",
        "updated_at": "2020-07-06T12:48:49.591Z",
        "task_id": 0,
        "visible": true
    },
    {
        "id": 691465,
        "job_id": 123,
        "severity": 1,
        "severity_description": "INFO",
        "progname": "worker",
        "message": "No client definitions. Using common definitions.",
        "created_at": "2020-07-06T12:48:50.079Z",
        "updated_at": "2020-07-06T12:48:50.079Z",
        "task_id": 24100,
        "visible": true
    },
    ...
    ...
    ...
    {
        "id": 691516,
        "job_id": 123,
        "severity": 1,
        "severity_description": "INFO",
        "progname": "worker",
        "message": "'rename_manifests' completed successfully",
        "created_at": "2020-07-06T12:48:52.282Z",
        "updated_at": "2020-07-06T12:48:52.282Z",
        "task_id": 24102,
        "visible": true
    },
    {
```

(continues on next page)

(continued from previous page)

```
        "id": 691519,
        "job_id": 123,
        "severity": 1,
        "severity_description": "INFO",
        "progname": "controller",
        "message": "job has completed successfully",
        "created_at": "2020-07-06T12:48:52.684Z",
        "updated_at": "2020-07-06T12:48:52.684Z",
        "task_id": 24102,
        "visible": true
    },
    {
        "id": 691520,
        "job_id": 123,
        "severity": 1,
        "severity_description": "INFO",
        "progname": "callback",
        "message": "No client definitions. Using common definitions.",
        "created_at": "2020-07-06T12:48:53.421Z",
        "updated_at": "2020-07-06T12:48:53.421Z",
        "task_id": 24102,
        "visible": true
    },
]
```

400 - Error Response:

```
{
    "error": "<error message>"
}
```

SCHEDULER ENDPOINTS

GET: `/schedules`

Returns a list of the currently active schedules. More information about the Task Engine scheduler can be found here

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication

Optional Headers:None

Successful Response:

```
{
    "result": "ok",
    "schedules": {
        "queue_scheduled_jobs": {
            "class": "QueueJobs",
            "every": [
                60,
                {
                    "first_in": 5
```

(continues on next page)

(continued from previous page)

```
                }
            ],
            "queue": "scheduler",
            "description": "Enqueues scheduled jobs that have a run_at time in the␣
↪past."
        }
    }
}
```

400 - Error Response:

```
{
    "error": "<error message>"
}
```

PUT: `/scheduler`

This endpoint allows for activating or deactivating schedules. More information about the Task Engine scheduler can be found here

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication

Optional Headers:None

Payload parameters:

- `schedule` - name of the schedule to be activated or deactivated
- `active` - accepts true or false to set the schedule to active or inactive

Sample Payload:

```
{
    "schedule": "<schedule name>",
    "active": true
}
```

Successful Response:

```
{
    "result": "ok",
    "schedules": ["<list of schedules>"]
}
```

400 - Error Response

```
{
    "error": "<error message>"
}
```

SETTINGS ENDPOINTS

POST: `/settings`

This settings endpoint is used to update or create new Task Engine settings. Only one setting can be added or updated at a time.

System default settings:

- `max_jobs` - The maximum number of concurrent jobs. Default: 2
- `priority_slots` - The number of concurrent job slots that should be reserved for high priority jobs. More information can be found here. Default: 0
- `priority_threshold` - The threshold at which jobs will start being considered as priority. Default: 5.
- `schedule_interval` - The interval, in seconds, between scheduler executions. Default: 60
- `retry_delay` - The delay, in seconds, between retries for failed Resque tasks. Default: 5
- `retry_limit` - The number of times a Resque task should be retried before a job is abandoned. Default: 3

Requires Authentication: Yes Required Headers:

- `client` - client name, required for authentication
- `api-key` - required for authentication
- `content-type` - set to `application/json`

Optional Headers:None

Payload parameters:

- `name` - setting name from the list above or name for a new setting
- `setting` - the value to be given to that setting

Sample Payload:

```
{
    "name": "max_jobs", // setting name
    "setting": "4" // value
}
```

Successful Response:

```
{
    "result": "ok",
    "message": "<setting name> setting created/updated"
}
```

400 - Error Response

```
{
    "error": "<error message>"
}
```

2.1.3 TASK ENGINE WORKFLOW FEATURES

PRIORITY

The Task Engine supports ordering of jobs by priority. The priority parameter can be submitted as part of the json payload being submitted. The priority is in ascending order as follows:

```
1 - Top Priority
.
.
5 - Default
.
.
10 - Least Priority
```

The "priority" parameter needs to be submitted within the "job" section of the json payload as shown below:

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodcapture",
    "priority": 3
  },
  "parameters": {
    "content_id": "demo1",
    ...
    ...
    ...
  }
}
```

Whenever an execution slot is available, the system will first check by priority and then check the submission time and date of the job. In the case where multiple jobs are executed with the same priority (eg. with the default priority 5), the Task Engine operates in a FIFO (First In First Out) manner.

Priority Slots

Two settings are available to further enhance support for priority jobs.

- Priority Slots - The number of job slots reserved for priority jobs.
- Priority Threshold - The priority at which a job can run within a priority slot. The threshold value defaults to `5` and cannot be less than `1`.

The advantage of using priority slots is to stop the queue from being held up by low priority jobs. This is especially useful if long running jobs are given a lower priority as they can be queued up without exhausting the setup's concurrency availability. It is also a good way of fast tracking certain types of jobs by giving them a higher priority and setting the threshold to an appropriate value. This ensures that the priority slots are reserved for such jobs.

The slots and threshold can be modified on the fly through the Task Engine API settings endpoint.

Important note: This will not increase the number of max concurrent jobs but it will reserve some fo the concurrency for jobs with priority between 1 and the `Priority Threshold`. As an example, if a setup has 5 maximum concurrent jobs and the priority slots is set to 2, any job can ustilise 3 concurrency slots but only priority jobs can utilise the 2 priority slots.

STITCHING CLIPS

The Task Engine includes a feature that will allow multiple clips to be stitched together into a single clip, in a single job. This can be done by defining multiple objects within the "clips" parameter in the json payload for VOD Capture. This also allows a mixture of live and VoD sources to be captured and stitched together into a new clip. The example below shows how the "clips" parameter would need to be provided to achieve this.

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodcapture"
  },
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "source": "http://mydomain.com/copyright.ism/manifest"
      },
      {
        "source": "http://mydomain.com/live.isml/manifest",
        "start": "2018-06-06T10:00:00.000",
        "end": "2018-06-06T10:30:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
      },
      {
        "source": "http://mydomain.com/live.isml/manifest",
        "start": "2018-06-06T10:35:00.000",
        "end": "2018-06-06T11:00:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
      }
    ],
    ...
    ...
    ...
  }
}
```

MULTIPLE SOURCES

In some cases, a live stream could have multiple origins setup (eg. for load balancing the origin servers). The Task Engine, allows for both streams to be defined as the source for a capture. It is smart enough to find which live stream will provide the best output capture and use that stream as the source. If the Task Engine discovers discontinuities within the streams, it will use segments from both streams to try and generate a clip with the least number of missing fragments.

The streams can be defined in the `"sources"` parameter when executing the VOD Capture workflow.

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodcapture"
  },
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "sources": [
          "http://mydomain.com/live_1.isml/manifest",
          "http://mydomain.com/live_2.isml/manifest"
        ],
        "start": "2018-06-06T10:00:00.000",
```

(continues on next page)

(continued from previous page)

```
        "end": "2018-06-06T10:30:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
      }
    ],
    ...
    ...
    ...
  }
}
```

In this case, `"sources"` replaces the `"source"` parameter, however; it can still be used in conjunction with other clips which only contain a single stream as shown below.

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodcapture"
  },
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "source": "http://mydomain.com/copyright.ism/manifest",
      },
      {
        "sources": [
          "http://mydomain.com/live_1.isml/manifest",
          "http://mydomain.com/live_2.isml/manifest"
        ],
        "start": "2018-06-06T10:00:00.000",
        "end": "2018-06-06T10:30:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000"
      }
    ],
    ...
    ...
    ...
  }
}
```

GENERATE DOWNLOAD CLIPS

The Task Engine VOD Capture workflow supports generating download clips without creating VoD assets. This is done by setting the property `"generate_vod"` to false and `"generate_mp4"` to true. It is important that if `"generate_vod"` is set to false, to not manually override the `"create_dref"` parameter. Setting `"create_dref"` to true will lead to a failed workflow as this requires VoD assets to generate DREF mp4s.

The resulting download will be an MP4 containing all the video, audio and caption tracks defined using the clip's `"filter"` parameter. If no filter is defined, the resulting MP4 will contain all the tracks available in the stream.

SCHEDULER

The Task Engine supports scheduling of jobs via the `run_at` and `sempahore_url` job attributes. Jobs are moved from a queue_state of `scheduled` to a queue_state of `queued` via a scheduler-worker. The interval at which this runs is pulled from the database settings table (schedule_interval, default: 1 hour).

The scheduler-worker looks for jobs which have a queue_state of `scheduled`, a `run_at` time in the past and a `semphore_url` that returns a successful response (2XX/3XX).

The schedule_interval can be set via an api call. (where x is time in seconds)

`post '/settings'`

```
{
  "client": "demo-client",
  "name": "schedule_interval",
  "setting": "<x>"
}
```

A jobs `run_at` and `semaphore_url` attributes can be set in multiple ways. The `run_at` defaults to the time the job was created. If the job's `run_at` time is in the future, a log will be added to indicate such. The `run_at` time format should be `yyyy-MM-ddTHH:mm:ss.fff`.

1. When submitting a job

`post '/job'`

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodcapture",
    "run_at": "2040-06-06T10:00:00.000",
    "semaphore_url": "https://www.vualto.com/"
  }
}
```

In the above example the job will be queued at 10:00AM on the 6th of June 2040 and when `https://www.vualto.com/` returns a successful response. The following message `Job will run at: "2040-06-06T10:00:00.000"` will be logged against the job.

1. When updating an existing job

`put '/jobs/:job_id'`

```
{
  "client": "demo-client",
  "run_at": "2040-06-06T10:00:00.000",
  "semaphore_url": "https://www.vualto.com/"
}
```

1. When submitting a capture with a clip end time in the future

If a capture is submitted with a clip end time that is in the future, it will be automatically scheduled to run at the end time of the clip which is furthest in the future. The exception to this is if the `run_at` time is specified and is further in the future than the end time, then the `run_at` time will be used.

TRACK PROPERTIES

There are instances when track properties need to be added to specific tracks within the VOD manifest. This usually occurs when custom track descriptions or track roles need to be set. The Task Engine supports adding track properties to audio and subtitle tracks. Filtering of tracks is based on type (`audio` or `textstream`) and a combination of language and/or track role. The filters and values can be set in Vualto's Central Configuration so they can easily be applied to all VODs being captured or ingested. They can also be defined as part of the job submission. The value set will overwrite the existing value for the property, if it already exists. Below are some samples of how the filters can be defined.

Setting the defined track description where the audio language is not set or set to `und` (undefined).

```
"track_properties": {
  "audio": [
    {
      "language": "und",
      "properties": {
        "track_description": "Original Audio Track"
      }
    },
    {
      "language": "",
      "properties": {
        "track_description": "Original Audio Track"
      }
    },
  ]
}
```

Setting the defined track description and track name where the language is set `eng` and the role is set to `description`.

```
"track_properties": {
  "audio": [
    {
      "language": "eng",
      "role": "description",
      "properties": {
        "track_name": "Audio Description - English",
        "track_description": "English Audio Descriptive"
      }
    }
  ]
}
```

Setting the defined track role and description to the subtitle track where the language is set to `eng`.

```
"track_properties": {
  "textstream": [
    {
      "language": "eng"
      "properties": {
        "track_role": "caption",
        "track_description": "English CC"
      }
    }
  ]
}
```

Adding a track description to the audio description track.

```
"track_properties": {
  "audio": [
    {
      "language": "eng",
```

(continues on next page)

(continued from previous page)

```
      "kind": "main-desc",
      "properties": {
        "track_description": "English (describes video)"
      }
    }
  ]
}
```

Setting a combination of properties to both audio and subtitle tracks.

```
"track_properties": {
  "audio": [
    {
      "language": "und",
      "properties": {
        "track_description": "Original Audio"
      }
    },
    {
      "language": "",
      "properties": {
        "track_description": "Original Audio"
      }
    },
    {
      "language": "eng",
      "role": "alternate"
      "properties": {
        "track_name": "English Alt",
        "track_description": "English Alternate track"
      }
    },
    {
      "language": "eng",
      "properties": {
        "track_role": "main"
      }
    }
  ],
  "textstream": [
    {
      "language": "eng",
      "properties": {
        "track_role" : "caption",
        "track_description": "English CC"
      }
    }
  ]
}
```

Supported properties are:

- `track_name`
- `track_description`
- `track_role`
- `track_language`

Supported filters are:

- `language` (retrieved from the .ism)
- `role` (retrieved from the .ism, should not be mixed with `kind` in the same track)
- `kind` (retrieved from the track file, should not be mixed with `role` in the same track)

FILE PROPERTIES

File properties can be used to set the track kind and language in audio tracks. Track kind allows accessibility options, such as audio description tracks, to be indicated. The language of the track can also be changed using this property. This replaces the legacy behaviour of specifying the language between the last underscore and the extension (`_<language>.m4a`).

Here is the list of available track kinds.

Here is an example for marking audio1.m4a as the Dutch audio description track:

```
"file_properties": {
  "audio1.m4a": {
    "kind": "main-desc",
    "language": "dut",
  }
}
```

Both `kind` and `language` are optional. If `language` is specified, then this will override the legacy behaviour of checking the language in the file name.

STORAGE SUPPORT

The Task Engine supports multiple storage types for ingesting content and saving VOD. Support has also been added so a combination of storage types can be used for the same job. This can be done by setting the `source_storage` and `destination_storage` in the job payload (for supported workflows). Eg. Ingesting content from local storage and save VOD assets on S3 would require `source_storage` to be set to `local` and `destination_storage` to be set to `S3`.

The system default storage type is Amazon S3, however; the default can be customised per client as well as set on a job per job basis. Additional setup may be required when using `local` storage, as the folders will need to be mapped to the worker docker containers.

Natively supported storage types:

- Amazon S3 (`S3`)
- Azure Blob Storage (`azure_blob`)
- On premises infrastructure (`local`)

PREVIEW THUMBNAILS

Note:

Trickplay support has been added to the VOD Capture and VOD Stream workflows. This can be used as an alternative to generating the preview thumbnails.

Preview thumbnails refers to the thumbnails that appear on the a video player's timeline as the user hovers over the progress bar. These can be generated on 3 different occasions:

- When capturing content, by setting the `preview_thumbnails` property to `true` when submitting a VOD Capture job.
- When ingesting content, by setting the `preview_thumbnails` property to `true` when submitting a VOD Stream job.
- By submitting a separate Build Thumbnails job.

The process is pretty much the same for all of the above. A sprite is generated with thumbnails at every 'x' second intervals (default is 10 seconds) and a VTT file which relates each thumbnail within the sprite to corresponding time within the stream. These files must be made accessible to the players. On cloud platforms, this is usually done by create a CDN for .jgp and .vtt files. The Vualto Assets API can be used to retrieve the URL for the VTT file.

Important Note: The interval is a very close approximation and not an exact interval.The thumbnail image is generated from the closest iframe.

Different players reference the VTT file differently. The following are some examples of how some players reference them:

- Bitmovin
- THEOPlayer

AVOD AND LIVE COMPOSE

The Vualto Task Engine now supports creating AVOD and Live Compose playlists using the vodremix workflow.

- AVOD refers to a playlist of clips (VOD or MP4s) with support for server side ad insertion and replacement through SCTE35 markers.
- Live Compose refers to a playlist of clips (VOD or MP4s), played out as a live stream, with support for server side ad replacement through SCTE35 markers. Ad insertion is currently not supported for simulated live streams. Live Compose playlists can be created by setting the `live_compose` flag to `true` in the job payload. To set a specific start time for a live stream use the `stream_start_time` parameter. If this is not set, the live stream will begin as soon as the manifest is packaged and available. The rest of the payload is identical to AVOD.

A markers object can be added to each clip object. The marker object supports setting the timescale and sync samples for each marker. The main use for the marker object is to specify meta events (SCTE35 markers. It's important to note that the `presentation_time` property of each meta event is relative to the clip and will always consider the clip start to be "00:00:00".

When adding meta events for ad replacement (`"type":  "replace"` - default) the `duration` property will indicate how much of the original clip content is to be replaced by an ad. When adding meta events for ad insertion (`"type":  "insert"`) the `duration` property indicates how long the ad inserted will be. The original clip content will continue from where it stopped when the ad is inserted.

The AVOD example below shows three clips being stitched together. The first clip has two SCTE 35 markers, a 4 minute and 30 second replacement marker at the 10 minute mark and a 30second ad insertion at the 16 minute mark. The second clip doesn't have any SCTE 35 markers. The third clip has a 2 minute replacement marker at the 5 minute mark.

```
{
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "sources": [
          "https://bucket.s3-eu-west-1.amazonaws.com/clip1_1080.mp4",
```

(continues on next page)

(continued from previous page)

```
          "https://bucket.s3-eu-west-1.amazonaws.com/clip1_720.mp4",
          "https://bucket.s3-eu-west-1.amazonaws.com/clip1_480.mp4"
        ],
        "markers": {
          "frame_accurate": true,
          "meta_events": [
            {
              "presentation_time": "00:10:00",
              "duration": "00:04:30.000"
            },
            {
              "type": "insert",
              "presentation_time": "00:16:00",
              "duration": "00:00:30.000"
            }
          ]
        }
      },
      {
        "sources": [
          "https://bucket.s3-eu-west-1.amazonaws.com/clip2_1080.mp4",
          "https://bucket.s3-eu-west-1.amazonaws.com/clip2_720.mp4",
          "https://bucket.s3-eu-west-1.amazonaws.com/clip2_480.mp4"
        ],
        "start": "00:00:00.000",
        "end": "00:10:00.000",
        "frame_accurate": "true"
      },
      {
        "sources": [
          "https://bucket.s3-eu-west-1.amazonaws.com/clip3_1080.mp4",
          "https://bucket.s3-eu-west-1.amazonaws.com/clip3_720.mp4",
          "https://bucket.s3-eu-west-1.amazonaws.com/clip3_480.mp4"
        ],
        "start": "00:00:00.000",
        "end": "00:20:00.000",
        "frame_accurate": "true",
        "markers": {
          "meta_events": [
            {
              "type": "replace",
              "presentation_time": "00:05:00",
              "duration": "00:02:00"
            }
          ]
        }
      }
    ],
    "dvr_window_length": 60,
    "drm": [
        "fairplay",
        "playready",
        "cenc",
        "widevine"
    ],
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
```

(continues on next page)

(continued from previous page)

```
      "http://your.custom.endpoint"
    ],
    "output_file": "remix.mp4"
  },
  "client": "demo-client",
  "job": {
      "workflow": "vodremix"
  }
}
```

LIVE COMPOSE WITH MANIFEST MANIPULATION

The VUALTO Task Engine now supports manifest manipulation to generate LIVE COMPOSE streams (VOD playlist looping), using AWS Mediatailor Channel Assembly. This workflow takes VOD streaming URLs directly and the resulting live stream fragments come from the original VOD streaming URLs - this could result in cost savings in caching layers.

With this workflow, it is also possible to condition live streams for SSAI (server side ad insertion), however, it requires ad slate clips (also in the form of VODs) in order to signal ad breaks. The slate clips are stitched linearly with the clip sources at the given `presentation_time` (relative to the clip source) and the relevant ad break timed metadata added to the resulting live stream. If no `presentation_time` is provided, it will default to `00:00:00` (pre-roll).

> Important! VOD sources must be encoded similarly. For example the same number of renditions, codecs, resolutions, etc. Job requests with mixed encoding profiles will fail validation.

The example below would result in a live stream where assets `source_1.m3u8`, `source_2.m3u8`, and `source_3.m3u8` would loop infinitely with ad breaks (where the ad content is ad-slate.m3u8) in between each clip. `source_3.m3u8` would have an additional ad break approximately 30 seconds in the video (mid-roll).

```
{
  "client": "demo-client",
  "job": {
    "workflow": "mediatailor_channel_assembly"
  },
  "parameters": {
    "content_id": "demo-content",
    "clips": [
      {
        "source": "https://cdn.com/assets/source_1.m3u8",
        "markers": {
          "meta_events": [
            {
              "slate": "https://cdn.com/assets/ad-slate.m3u8",
              "presentation_time": "00:00:00"
            }
          ]
        }
      },
      {
        "source": "https://cdn.com/assets/source_2.m3u8",
        "markers": {
          "meta_events": [
            {
              "slate": "https://cdn.com/assets/ad-slate.m3u8",
              "presentation_time": "00:00:00"
```

(continues on next page)

(continued from previous page)

```
              }
            ]
          }
        },
        {
          "source": "https://cdn.com/assets/source_3.m3u8",
          "markers": {
            "meta_events": [
              {
                "slate": "https://cdn.com/assets/ad-slate.m3u8",
                "presentation_time": "00:00:00"
              },
              {
                "slate": "https://cdn.com/assets/ad-slate.m3u8",
                "presentation_time": "00:00:30"
              }
            ]
          }
        }
      ],
      "rest_endpoints": [
        "http://your.custom.endpoint"
      ]
    }
}
```

CONTINUOUS CAPTURE

The Task Engine VOD Capture workflow supports the `continuous_capture` parameter which will begin capturing clips before the specified end time. By default it will begin capturing once 80% of the time has elapsed between the earliest start time of the specified clips and the latest end time of all clips, or if the difference is less than 60 seconds it will begin capturing at the earliest start time. This can be overridden by setting the `run_at` parameter.

2.1.4 TASK ENGINE WORKFLOWS

VOD STREAM

This workflow will generate a VOD asset from an offline source (eg. MP4). A server side manifest is created, with and/or without DRM, that can be used for on the fly delivery of VOD content via the Unified Streaming Platform.

VOD Stream: Parameters

VOD Stream: JSON Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodstream"
  },
  "parameters": {
    "content_id": "demo1",
```

(continues on next page)

(continued from previous page)

```
    "source_folder": "mz-ast-2055fcff-8cca-4e37-85b9-9647dbe50398-1",
    "delete_source": false,
    "enable_drm": true,
    "output_folder": "mz-ast-2055fcff-8cca-4e37-85b9-9647dbe50398-1",
    "drm": [
      "fairplay",
      "playready",
      "widevine"
    ],
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://aaa.com/end",
      "http://bbb.com/end"
    ],
    "create_thumbnail": true,
    "thumbnail_time": "1:34.000",
    "generate_mp4": true,
    "mp4_filename": "demo_sample.mp4",
    "mezzanine": true,
    "combine_sources": true,
    "create_dref": true,
    "all_audio_tracks": true,
    "preview_thumbnails": true,
    "preview_thumbnails_interval": 20,
    "apply_track_properties": true,
    "source_storage": "local",
    "destination_storage": "S3"
  }
}
```

VOD Stream: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

VOD CAPTURE

This workflow allows you to create a frame accurate VOD clip by passing in a start and end time. If the source stream contains time stamps, UTC time stamps can be used for the start and end times. The result will be a new VOD asset and/or a downloadable MP4.

VOD Capture: Parameters

VOD Capture: JSON Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodcapture"
  },
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "clips": [
      {
        "source": "http://mydomain.com/live.isml/manifest",
        "start": "2018-06-06T10:00:00.000",
        "end": "2018-06-06T10:30:00.000",
        "filter": "type==\"audio\"||type==\"video\"&&systemBitrate==1300000",
        "key_id": "346AS5847333DDSHKFSDS7633429CD33",
        "content_key": "346AS5847333DDSHKFSDS7633429CD33"
      }
    ],
    "enable_drm": false,
    "drm": [
      "fairplay",
      "playready",
      "cenc",
      "widevine",
      "aes"
    ],
    "frame_accurate": true,
    "transcode_proxy": "https://vualto.transcode-proxy.com",
    "copy_ts": false,
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://your.custom.endpoint"
    ],
    "create_thumbnail": true,
    "thumbnail_time": "1:34.000",
    "generate_mp4": true,
    "mp4_filename": "demo_sample.mp4",
    "create_dref": true,
    "preview_thumbnails": true,
    "preview_thumbnails_interval": 20
  }
}
```

VOD Capture: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

VOD DELETE

This workflow allows you to a delete VOD asset from storage.

VOD Delete: Parameters

VOD Delete: JSON Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "voddelete"
  },
  "parameters": {
    "content_id": "demo1",
    "folder": "vualto-test-1",
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://your.custom.endpoint"
    ]
  }
}
```

VOD Delete: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

DRM SWITCH

This workflow allows you to toggle DRM on and off for a VOD asset. Missing manifests will be generated when required. If the VOD asset does not have a DRM manifest and DRM is being enabled, a list of DRM systems needs to be provided as part of the payload.

DRM Switch: Parameters

DRM Switch: Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "drmswitch"
  },
  "parameters": {
    "content_id": "demo1",
    "folder": "vualto-test-1",
    "drm": [
      "fairplay",
      "cenc"
    ],
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://your.custom.endpoint"
    ]
  }
}
```

DRM Switch: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

CREATE MP4

This workflow allows you to create an MP4 from a VOD asset.

Create MP4: Parameters

Create MP4: Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "createmp4"
  },
  "parameters": {
    "content_id": "demo1",
    "source_folder": "vualto-test-1",
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
```

(continues on next page)

(continued from previous page)

```
      "http://your.custom.endpoint"
    ],
    "mp4_filename": "result.mp4",
    "output_folder": "vualto-test-1/downloads"
  }
}
```

Create MP4: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

BUILD THUMBNAILS

This workflow allows you to generate thumbnail assets which can then be used for video timeline previews.

Build Thumbnails: Parameters

Build Thumbnails: Payload example

```
{
    "parameters": {
        "content_id": "demo1",
        "source": "http://mydomain.com/example.ism/.m3u8",
        "output_folder": "vualto-test-1/downloads",
        "target_filename": "demo_sample",
        "preview_thumbnails_interval": 20,
        "video_fps": 24,
        "rest_endpoints": []
    },
    "client": "demo-client",
    "job": {
        "workflow": "build_thumbnails"
    }
}
```

Build Thumbnails: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

VOD REMIX

This workflow allows you to create a virtual VOD asset that is just a playlist referencing other VOD streams or video files.

VOD Remix: Parameters

VOD Remix: JSON Payload example

```
{
  "parameters": {
    "content_id": "demo_1",
    "output_folder": "demo_1",
    "transcode_proxy": "https://vualto.transcode-proxy.com",
    "clips": [
      {
        "source": "https://bucket.s3-eu-west-1.amazonaws.com/manifest.ism",
        "start": "2018-06-06T10:00:00.000",
        "end": "2018-06-06T10:30:00.000"
      },
      {
        "source": "https://bucket.s3-eu-west-1.amazonaws.com/ad.mp4",
        "output_description": true
      },
      {
        "source": "https://bucket.s3-eu-west-1.amazonaws.com/manifest.ism",
        "start": "2018-06-06T10:40:00.000",
        "end": "2018-06-06T11:00:00.000"
      },
      {
        "source": "https://bucket.s3-eu-west-1.amazonaws.com/manifest.ism",
        "start": "2018-06-06T11:00:00.000",
        "end": "2018-06-06T11:10:00.000",
        "frame_accurate": true,
      }
    ],
    "drm": [
        "fairplay",
        "playready",
        "cenc",
        "widevine",
        "aes"
    ],
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://your.custom.endpoint"
    ],
    "output_file": "remix.mp4"
  },
  "client": "demo-client",
```

(continues on next page)

(continued from previous page)

```
  "job": {
      "workflow": "vodremix"
  }
}
```

VOD Remix: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

GENERATE GIF

This workflow allows you to create animated GIFs from a VOD stream.

Generate GIF: Parameters

Generate GIF: Payload example

```
{
  "client": "demo-client",
  "job": {
      "workflow": "generate_gif"
  },
  "parameters": {
      "content_id": "demo-content",
      "source": "http://mydomain.com/example.ism/.m3u8",
      "start_at": 30,
      "duration": 6,
      "output_folder": "/demo-content/assets",
      "gif_filename": "half speed.gif",
      "bitrate": 1549288,
      "fps": 15,
      "width": 500,
      "playback_speed": 0.5,
      "rest_endpoints": [
        "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback
↪",
        "http://your.custom.endpoint"
      ]
  }
}
```

Generate GIF: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

CAPTURE FRAME

This workflow allows you to capture a single frame from a stream.

Capture Frame: Parameters

Capture Frame: Payload example

```
{
  "client": "demo-client",
  "job": {
      "workflow": "capture_frame"
  },
  "parameters": {
      "content_id": "demo-content",
      "source": "http://mydomain.com/example.ism/.m3u8",
      "frame_time": "00:00:07.0400000",
      "output_folder": "/demo-content/assets",
      "image_filename": "frame.jpg",
      "rest_endpoints": [
        "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback
↪",
        "http://your.custom.endpoint"
      ]
  }
}
```

Capture Frame: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

ASSET DELETE

This workflow allows you to delete individual assets without deleting an entire VOD directory.

Asset Delete: Parameters

Asset Delete: Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "asset_delete"
  },
  "parameters": {
    "content_id": "demo-content",
    "files": [
      "demo-content/assets/foo.gif",
      "demo-content/thumbnail.jpg"
    ],
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://your.custom.endpoint"
    ]
  }
}
```

Asset Delete: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

MEDIATAILOR CHANNEL ASSEMBLY

This workflow allows for creating and updating Live Compose streams - very similar to VOD REMIX Live Compose. Please refer to Live Compose with Manifest Manipulation for more information and ad break signaling examples.

MediaTailor Channel Assembly: Parameters

MediaTailor Channel Assembly: Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "mediatailor_channel_assembly"
  },
  "parameters": {
    "content_id": "demo-content",
    "clips": [
      {
        "source": "https://cdn.com/assets/1.m3u8"
      },
      {
        "source": "https://cdn.com/assets/2.m3u8"
      },
      {
        "source": "https://cdn.com/assets/3.m3u8"
      }
    ],
    "rest_endpoints": [
      "http://your.custom.endpoint"
    ]
  }
}
```

MediaTailor Channel Assembly: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

MEDIATAILOR CHANNEL STATE

This workflow allows for stopping and starting MediaTailor channels.

MediaTailor Channel State: Parameters

MediaTailor Channel State: Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "mediatailor_channel_state"
  },
  "parameters": {
    "content_id": "demo-content",
```

(continues on next page)

(continued from previous page)

```
    "state": "stop",
    "delete": true
  }
}
```

MediaTailor Channel State: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

VOD NPVR

This workflow will generate a VOD asset from segments captured through the Vualto Archiver. Segments are shared across different VOD assets which reduces storage requirements and processing time.

A server side manifest is created, with and/or without DRM, that can be used for on the fly delivery of VOD content via the Unified Streaming Platform. The VOD NPVR workflow includes support to inherit the DRM keys form a specified Vualto Archiver profile and stores it as a custom manifest. This workflow will include any SCTE35 markers that occurred during each segment.

VOD NPVR: Parameters

VOD NPVR: JSON Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "vodnpvr"
  },
  "parameters": {
    "content_id": "vudrm_1",
    "clips": [
        {
            "start": "2020-09-08T14:10:00Z",
            "end": "2020-09-08T14:44:00Z",
            "capture_id": "test4"
        }
    ],
    "transcode_proxy": "https://vualto.transcode-proxy.com",
    "output_root": "output_root",
    "apply_custom_drm": true,
    "profile_id": "test4_drm",
    "custom_manifest_name": "manifest.ism",
```

(continues on next page)

(continued from previous page)

```
    "drm": [
      "fairplay",
      "cenc",
      "clear"
    ],
    "cpix": false,
    "missing_content_limit": 700,
    "output_folder": "vudrm/test4_1599574200000_1599576240000",
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://aaa.com/end",
      "http://bbb.com/end"
    ],
    "custom_data": { "custom_ref" : "ref-123" }
  }
}
```

VOD NPVR: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

Trickplay

This workflow will generate (with `trickplay_thumbnails` enabled) a thumbnail CMAF track containing JPEG compressed frames from points in a AVC/H.264 or HEVC/H.265 video.

If `trickplay_thumbnails` is disabled, it will only insert sync-samples.

Trickplay: Parameters

Trickplay: JSON Payload example

```
{
  "client": "demo-client",
  "job": {
    "workflow": "trickplay"
  },
  "parameters": {
    "content_id": "vudrm_1",
    "source_folder": "vualto-test-1",
    "trickplay_thumbnails": true,
    "trickplay_thumbnail_interval": 5,
```

(continues on next page)

(continued from previous page)

```
    "trickplay_thumbnail_size": 1280,
    "trickplay_thumbnail_quality": 50,
    "output_root": "output_root",
    "output_folder": "vualto-output",
    "rest_endpoints": [
      "https://vis.vuworkflow.staging.vualto.com/api/event/vuflow/taskenginecallback",
      "http://aaa.com/end",
      "http://bbb.com/end"
    ],
    "custom_data": { "custom_ref" : "ref-123" }
  }
}
```

Trickplay: Callback properties

Task Callback

Task callbacks are triggered after each task within a workflow is completed. Below is a list of the default properties for the callback:

Job Callback

Job callbacks are triggered when the entire job has completed. Below is a list of the default properties for the callback.

WORKFLOW TRIGGER EXAMPLE

Example of a curl command to trigger ingest for the *VOD Stream* workflow:

```
curl -X POST \
http://vualto.demo.com/job \
-H "API-KEY: aabbccdd-1122-3344-5566-eeff77889900" \
-H 'Cache-Control: no-cache' \
-H 'Content-Type: application/json' \
-d '{
  "client": "vualto",
  "job": {
    "workflow": "vodstream"
  },
  "parameters": {
      "content_id": "demo_1",
      "source_folder": "/input/demo1",
      "delete_source": false,
      "enable_drm": false,
      "output_folder": "/test",
      "drm": [
        "fairplay",
        "playready",
        "cenc",
        "widevine"
      ],"rest_endpoints": [
        "https://webhook.site/55151d14-cee1-416b-b956-a90525ae8f58",
        "https://webhook.site/bc4c13ee-f118-4d5b-a4af-7ac07890a7f1"
```

(continues on next page)

(continued from previous page)

```
      ],
      "create_thumbnail": false,
      "generate_mp4": true,
      "combine_sources": true,
      "create_dref": true,
      "all_audio_tracks": false,
    }
  }
```

This results in the files `<content_id>_<drm_tag>_<unique_guid>.ism` and `<content_id>_<unique_guid>.ismv` being produced in the folder:

`<configured_root>(/<optional_output_folder>)/<content_id>`

The response from this call should be either a 200 OK, with the following payload:

`{ "job_id": <job_id>, "result": "accepted" }`

or a 400 BAD REQUEST with the following payload:

`{ "error": "<description_of_error>" }`

Assuming the call is successful, this would add an ingest job to the Task Engine queue, with the files to be ingested expected to be in the following location:

`<input_root>/input/demo1`

If the process completes successfully, then the output would be the following files:

`<output_root>/test/demo1.ism` `<output_root>/test/demo1.ismv`

If the 'output_folder' parameter was excluded, then the files would be output to the following locations:

`<output_root>/demo1.ism` `<output_root>/demo1.ismv`

NOTE: there may be some additional files, depending on the exact processes involved, but the minimum would usually be these.

2.2 RELEASE NOTES

- *v2.1.x*
- *v2.0.x*
- *v1.173.x*
- *v1.172.x*
- *v1.171.x*
- *v1.170.x*
- *v1.169.x*
- *v1.168.x*
- *v1.167.x*
- *v1.166.x*
- *v1.165.x*

2.2.1 v2.1.x

2.1.2 - 2023-07-05

- Fixed an issue where the incorrect duration was being used to decide the transcode options.
- Added support for USP 1.12.3

2.1.1 - 2023-05-16

- Fixed Twitter upload not working.
- Added the option to enabled/disable the use of a cpix document instead of the cpix proxy when generating VOD using the following workflows:
  - vodcapture
  - vodnpvr
  - vodremix
  - vodstream

2.1.0 - 2023-04-19

- Cleaned up FFmpeg errors.
- Added support for Linux based trickplay track generation.
- Fixed an issue in the `drmswitch` workflow were the files array did not return full paths.
- Workflows can now be setup for more concurrent task execution.

2.2.2 v2.0.x

2.0.5 - 2023-04-19

- Changed track properties format, the old format is deprecated but still works.
- Track properties can now match based on the track kind.

2.0.4 - 2023-03-31

- Adding an index to audio tracks to avoid accessibility audio tracks from being set as the default.

2.0.3 - 2023-03-27

- Removed + from generated filenames to avoid issues with Apache on VOD Origins

2.0.2 - 2023-03-22

- Fixed null objects being added to the json metadata file
- Added `file_parameters` property to `vodstream` to specify accessibility settings and language without needing to change the name of the file.

2.0.1 - 2023-03-13

- Added support for USP 1.12.1.
- Removed native support for USP versions older than 1.11.20.

2.0.0

- Added official multi-tenancy support.
  - Clients using a multi-tenant cluster will still have their own queue settings.
- Added support for client filtering in the queue page
- Client name is displayed in job rows.
- Added support for continuous captures.
  - This feature is experimental and needs to be explicitly specified when submitting a capture job.
  - This feature enables captures that end in the future to start before the end time and continuously capture until the end time is reached.
- Optimised job submission performance.
- Added support for `notify_subscribers` for YouTube syndication.
- Authentication has been made a requirement for more API endpoints.
- Removed native support for USP versions older than 1.11.12.
- The base images now run on top of Ubuntu 20
- A new persist flag has been added to jobs.
  - Jobs flagged to persist will not be cleared as part of the 30 day routines.
- Added a check on ingest (`vodstream`) workflows to fail the job as soon as no sources are found.
- General performance improvements.

2.2.3 v1.173.x

1.173.11 - 2022-12-05

- Hotfix: Fixed a race condition where a job may be started twice if multiple controllers are present

1.173.10 - 2022-11-14

- Hotfix: vodremix workflow no longer uss the default package options `--timed_metadata --splice_media`

1.173.9 - 2022-09-01

- Hotfix: NPVR hotfix to use archiver usp packaging options.
- Hotfix: NPVR hotfix to use symlinks when using local storage.
- Updated local storage:

  - Added support for symlinks.

1.173.8 - 2022-08-22

- Hotfix: NPVR hotfix to support nil values for segment end iframe times.

1.173.7 - 2022-08-02

- Hotfix: Deducting start time from the duration value when generating mp4s

1.173.6

- Patch: Added rescue to Twitter publications to handle forbidden actions.
- Patch: Changed the source used for trickplay thumbnail generation to improve performance.
- Patch: Update mediatailor_channel_assembly workflow.
  - Added support for updating an existing VOD sources.
- Hotfix: Fixed the duration by reducing the time value from the metadata. This was causing issues when assets included capture timestamps.
- Hotfix: Removed hardcoded no multiplex option when packaging DRM manifests that include FairPlay.

1.173.1

- Improved filtering for trickplay track selection.

1.173.0

- Bugfix: Fixed API error codes for issues when submitting a job with missing parameters.
- Added support for the latest USP GA release, 1.11.9.

Callbacks

- Added UTC date and time to all workflow callbacks
- Added the client name to all workflow callbacks

VOD Capture

- New `seed` parameter has been added to the clip object. This can be used instead of the `content_key` and `key_id` to capture from VUDRM encrypted streams.
- Frame Accuracy is automatically enabled for captures from streams with discontinuities. This reduces the chance of audio/video sync issues.
- Added support for adding trickplay to captures.

VOD Stream

- Added support for adding trickplay to ingests.

2.2.4 v1.172.x

1.172.5 - 2021-08-18

- Minor hot fixes introduced by 1.172.3 release

v1.172.3 - 2021-08-17

- Added support for transmuxing when generating the mp4 based on the duration of the content.

v1.172.0 - 2021-06-07

- Added `mediatailor_channel_assembly` workflow for creating and updating MediaTailor VOD playlist channels.
- Added `mediatailor_channel state` workflow for starting, stopping and deleting MediaTailor VOD playlist channels.
- Added support for the `priority_threshold` setting.
- Minor UI updates to reflect the new setting.

VOD Capture

- New `transcode_proxy` parameter for off loading frame accurate transcoding to a remote proxy.

VOD Remix

- New `stream_start_time` parameter for setting the stream start time for a Live Compose stream.
- New `dvr_window_length` parameter for setting the DVR window for Live Compose streams
- New `custom_active_manifest_name` parameter to allow the consumer to specify the manifest name for the resulting stream.
- New `transcode_proxy` parameter for off loading frame accurate transcoding to a remote proxy.

VOD Remix

- New `transcode_proxy` parameter for off loading frame accurate transcoding to a remote proxy.

2.2.5 v1.171.x

v1.171.1 - 2021-04-08

- Added a task to extract event metatdata to a json file.
- Added event duration to the job callback within a new metadata object.
- Bugfix: fixed an issue that could cause re-run job to fail.

VOD Stream

- Output transcoding step has been updated to only execute when an mp4 is requested.
- Added support for SRT subtitle files.
- Optimised the process of preparing captions for packaging.
- Extracting the audio track from the original source when transcoding the source.
- Added support for specifying the Bitmovin encoder version for transcoding.
- Added support for m4v files as encoding sources.

VOD Capture

- Output transcoding step has been updated to only execute when an mp4 is requested.
- Clip sorting for discontinuities has been improved.

VOD Remix

- Added support for creating AVOD playlists with SCTE35 ad insertion and replacement markers.
- Added support for creating live streams from VOD content with SCTE35 ad replacement markers.

Asset Delete

- Bugfix: Added checks for empty arrays before deleting attempting to delete files.

2.2.6 v1.170.x

v1.170.4 - 2020-11-18

- Bugfix: Fixed failing re-run job endpoint.

v1.170.3 - 2020-11-18

- Bugfix: Transcoding file extension check was case sensitive.

v1.170.2 - 2020-10-21

- Bugfix: Fixed issue causing some scheduled jobs to never be queued.

v1.170.1 - 2020-09-30

- File properties in callbacks are returned as a JSON array instead of a string containing an array.
- A new `custom_data` parameter has been added to all workflows to allow consumers to submit references they want returned in the final job callback.
- Running jobs can be paused using the update job endpoint and setting the queue_state to paused.
- SSL certificate verification of callbacks can be disabled.

VOD Stream

- Bugfix: The filename for audio tracks was not being checked for a language code when the language is missing within the metadata.

VOD Capture

- Stream decryption keys (`key_id` and `content_key`) now need to be specified as part of the clip object. This allows for the capture and stitching from two streams encrypted with different keys.
- Native support for capturing from local ingested streams through a new `local_source` property within clip objects.

VOD Delete

- The `content_id` parameter is now optional.

DRM Switch

- For VUDRM clients, if a clear or DRMed manifest is missing when trying to switch DRM on or off, the manifest will be generated dynamically.

VOD NPVR

- New workflow for capturing VOD from Vualto archiver segments.
- This workflow is intelligent in that a segment is only uploaded once even if it is used by multiple VODs.
- SCTE35 markers are preserved and included within the VOD.
- Supports custom manifests that apply DRM keys from Vualto Archiver profiles to the resulting VOD.

2.2.7 v1.169.x

v1.169.3 - 2020-07-09

- Dashboard hotfix.

v1.169.2 - 2020-07-07

- General release build.

v1.169.1 - 2020-07-07

- Build fixes.

v1.169.0 - 2020-07-06

- API enhancements for UI integrations.
- Updated to a newer Ruby version.
- Support for specifying Bitmovin encoding region.
- Bitmovin encodings now have client labels assigned.
- Queue reservation functionality for priority jobs.
- Optimised thumbnail and VTT generation for timeline preview thumbnails.

2.2.8 v1.168.x

v1.168.1 - 2020-03-17

- Added workflows for:
  - generating gifs
  - capturing single frames as jpgs
  - deleting multiple assets without deleting an entire VOD.
- Added support for multiple file formats when ingesting and encoding using Bitmovin.
  - Full list of supported source files are:
    * mp4
    * mov
    * mpg
    * mkv
    * avi
- Added number of failed jobs in the health check endpoint response.
- Bugfix: Added index to audio tracks when ingesting mp4s. This is to ensure each track name is unique.
- Added support for USP 1.10.18.

2.2.9 v1.167.x

v1.167.2 - 2019-11-04

- Bitmovin enhancements for re-packaging.

v1.167.1 - 2019-10-29

- Bitmovin integration now supports setting the ACL permission for the mp4 outputs.

v1.167.0 - 2019-10-25

- Added support for Azure Blob Storage as a source and/or destination storage option.
  - The option needs to be specified as `azure_blob`.
- Added the functionality to encode ingest source into multiple smaller resolutions.
  - Caveat: Currently this only works if there is a single MP4 source in the ingest folder.
- Task Engine version is now visible within the queue page and returned as part of the health check api request.
- Updates to the preview thumbnail (trickplay) generation to improve performance.
- Updated VOD remix:
  - Added support for Frame Accurate clips.
  - Added option for setting the profile (for a clip) for the remix output.
- Updated Task Engine queue and logs UI.
  - Completed status filter will no longer load failed jobs.
  - Search by job IDs is now supported. Multiple job IDs must be comma separated.
  - There is no longer a requirement for both From and To date to be provided for the date filter to work.
  - Search results will now include the Created At information for the jobs.
  - Word wrapping has been added to logs.
- Added `output_root` option to all workflows.
- API error codes updated.
- CPIX support for DRM packaging.
- Completed vodremix workflow.

2.2.10 v1.166.x

v1.166.8 - 2019-09-04

- Hotfix: Updated DRM switch callback message.

v1.166.7 - 2019-08-20

- Hotfix: supporting spaces in file inputs/outputs.

v1.166.6 - 2019-08-14

- Added support for USP 1.10.12.
- Removed support for USP 1.8.5.

v1.166.5 - 2019-08-13

- Added visible field to log with show_all param toggling.
- Improvements to the `build_thumbnails` workflow.
- Complete re-write of the way storage is handled:
  - Storage controller dictates which storage medium is used.
  - S3 storage re-written to support this.
  - Added local storage support.
  - Added support for different source and destination storage types within the same job.
    - Eg. Source for ingest can be a local MP4 and the destination for the packaged content is S3.
- Rewrite of DRM switch.
- Added vodcapture.json optional parameter "output_root" to specify the output root key.
- Added vodcapture.json optional parameter "empty_target" to specify whether to delete the contents of the output folder or not.
- New versioning system for the Task Engine.

2.2.11 v1.165.x

v1.165.0 - 2019-02-15

- Fixed issue with retry job call from the front end (queue page).
- Fixed bug with job scheduler not assigning the correct run_at time.
- Added a health check endpoint for improved Zabbix monitoring.
- Added proper support for TransDRM (without subtitles).
- Creating a clear manifest now requires the `clear` option to be added to the DRM list.
- Added `build_thumbnails` workflow (builds thumbnail sprite and vtt file for VOD content).
- Added transcoding option back to mp4 generation due to compatibility issues with Twitter.
- Added support for applying track properties to manifest files.
  - These can be added either through the job JSON payload or saved in Central Configuration.

2.3 SUPPORT

2.3.1 HOW TO CONTACT US

For all non-critical support requests please email support@vualto.com. If you are a registered user you can contact support via the help centre.

For all critical issues please contact support via one of the numbers below.

```
+44 (0)800 0314391
+44 (0)1752 916051
(800) 857 1808 (toll-free US number)
```

If there is a problem, please include as much information about the issue as possible.

2.3.2 SYSTEM REQUIREMENTS

The Task Engine is designed to run in Docker containers, and is therefore able to run on any Docker-supported Linux host. Recent releases of the Task Engine are also compatible with Kubernetes deployments and are fully integrated with Rancher for easy deployment and upgrade management. Any workers that need to run on Windows are currently **not** containerized but are installed as a native Windows service. Windows workers are a requirement for frame accurate captures.

When a worker starts a task, it will generally need to copy one or more files to storage to perform its work in an efficient manner, so it's critical that hosts have sufficient disk space for processing to take place. This is particularly important when dealing with video transcoding or DRM encryption. A shared storage is also used to store any assets required during job execution and facilitate horizontal scaling of workers. Vualto will usually advise on disk space requirements once the workflow is fully defined and some representative test content has been processed.