Animate 3D REST API

Revisions

Alpha v1.0.0

Initial rest APIs

Alpha v1.0.1

Added model parameter to the /process API

Alpha v1.2.0

Added custom character end points /character

Alpha v1.2.1

Added the “sim” parameter to API 3: Start Video Processing

Alpha v1.2.2

Added the “camera” parameter to API 3: Start Video Processing

Alpha v1.3.0

Added webhook APIs

Alpha v1.4.0

Added footLockingMode parameter to the /process API
Added new minutesBalance API

Alpha v1.4.1

Added flag “createThumb” to /character/storeModel API

Alpha v1.5.0

Exposed mp4 render out parameters in /process API
Added Face Tracking parameter in /process API

Alpha v1.5.1

Added /videoInfo API

The Animate 3D REST API lets you convert videos into 3D animations without having to use the DeepMotion Web Portal. Instead you can upload, process, and download the resulting FBX/BVH animations directly from an external application like a web or desktop app.

Authentication

The Animate 3D REST API uses basic **HTTP Authentication** to keep your requests and data secure. To use the API you will need a **Client ID** and a **Client Secret** which are provided by DeepMotion. If you do not have these please contact DeepMotion Support or your sales representative.

To retrieve your API access token you need to add the following Authorization header to your token request:

```
Authorization: Basic Base64(<clientId>:<clientSecret>)
```

where the value of `<clientId>:<clientSecret>` is **base 64** encoded. For Example, if your Client ID is `1a2b` and your client Secret is `3c4d` then your authorization header should look like this:

```
Authorization: Basic MWEyYjozYzRk
```

where `MWEyYjozYzRk` is the base64 encoded value of `1a2b:3c4d`.

Note: Optionally it is possible to send a unique user identifier (via x-useruid HTTP(S) header) along with the authorization header . See example below:

Curl -v -H “Authorization: Basic MWEyYjozYzRk” -H “x-useruid: me.us@test.com” -X GET “{host}/auth”

API Endpoints

All Animate 3D API requests must be made against the following base URL using the HTTPS protocol and port:

```
Staging Environment:          https://petest.deepmotion.com:443
Production Environment:       (Contact DeepMotion)
```

API Reference

API 1: Get Access Token

Desc	Authenticate client credentials and returns a time limited session cookie to be used in the subsequent REST API calls. After the session expiration, this API needs to be called again to get a new session cookie
Method + URI	GET {host}/session/auth
Header(s)	Authorization: Basic Base64(<clientId>:<clientSecret>)

Request	
Response	Sample Response Header: set-cookie: dmsess=s%3AEsF23MoyDEq7tTWQM8KfA_wjKkSrOFwU.2fjJTfDP%2FT2BeA5DFenwOH4t8XzqZsbSc6M2mZwS%2BWg; Domain=.deepmotion.com; Path=/; Expires=Mon, 03 Aug 2020 13:36:26 GMT; HttpOnly (Note: **dmsess** is the session cookie. This cookie needs to be sent in all subsequent REST API calls. Sample Request Header for other API calls: cookie:dmsess=s%3AEsF23MoyDEq7tTWQM8KfA_wjKkSrOFwU.2fjJTfDP%2FT2BeA5DFenwOH4t8XzqZsbSc6M2mZwS%2BWg)

API 2: Upload Video

Desc	Retrieves a signed url to upload video
Method + URI	GET {host}/upload
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	Query parameters: <name>: video file name (optional) <resumable>: 0 or 1(default) returns resumable or regular signed url (optional)

Response	JSON object:
	{
	"url": signed url
	}
	
	After retrieving the url, actual video upload is required to that storage url. If 'resumable' option is set in the request, we need one POST and one subsequent PUT request, otherwise a single PUT request will do the job.
	
	POST request to url:
	<x-goog-resumable>: start (set in the request header)
	<location>: resumable url (set in the response header by server)
	
	Put request to resumable url/url:
	attach raw bytes of the video file in the request body.

API 3: Start Video Processing

Desc	Start processing video after file has been uploaded to the designated URL
Method + URI	POST {host}/process
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>

Request	POST body should include a JSON object:
	{
	“url”: <upload url>
	“processor”: <processor_id>
	“params": [<params>, ...]
	}
	
	<upload_url> should match url returned from GET /upload request
	
	<processor_id> specifies which processor to use to process the video file, must be one of the following:
	
	Processor Id | **Description**
	video2anim | Deepmotion video to animation processor
	
	<params> specifies additional parameters that will be passed to the specified processor, for example:
	"params": [
	"config=configDefault",formats=bvh,fbx,mp4,model=<modelId>]
	
	Additional important parameter: **sim**
	This physics simulation parameter needs more clarification. This parameter influences Pose Estimation result to improve it in some cases like body parts inter penetration etc. If we would like to turn this ON, add sim=1 OR add sim=0 to turn it OFF. If we don’t add this parameter, simulation is turned off by default.
	
	For camera behavior in output video generation (mp4 for now), the default value for the **camera** parameter is **render.camera=closeup** which always keeps the simulated character in the camera frame with maximum zoom possible. **render.camera=fixed** is the other value that keeps the camera stationary.

	Another new parameter is: **poseEstimation.footLockingMode** or simply **footLockingMode** ● This parameter value can be one of the below: ○ **auto** : default mode, automatic switching between locking and gliding modes of the foot, recommended for general cases ○ **always** : forced foot locking all the time. only used when Auto mode can not remove all the foot gliding unsired ○ **never** : forced to disable foot locking and character grounding. used when the motion is completely in the air or in the water and therefore neither foot locking nor character grounding is needed. ○ **grounding** : forced disabling foot locking, however character is still grounded. Only used when Auto mode prevents the desired foot gliding (i.e. during shuffling dances) in the motion or locks the foot for too long on the ground during fast and short foot/ground contacts (i.e. during sprints or jumps.) **Mp4 render out parameters:** 1. Please add this below parameter, if you would like to generate the mp4 with only animated character in a default background (and without the original video): **render.sbs=0** 2. To replace the default background with a solid color (for green screening etc.) **render.sbs=0** **render.bgColor=0,177,64,0** (RGBA color code in the range of 0-255 for each channel, please note, the last channel (alpha) value is not in effect) 3. To set a studio like background with a solid color tint **render.sbs=0** **render.backdrop=studio** **render.bgColor=0,177,64,0** 4. To enable character shadow **render.shadow=1** **Face Tracking parameter:** Enables face tracking along with body tracking **trackFace=1**

Response	JSON object:
	{
	  "rid": <request id>
	}

API 4: Poll for Job Status

Desc	Polls for real-time status of a given processing job
Method + URI	GET {host}/status/rid GET {host}/status/rid1,rid2,..,rid
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	Clients can request current status of previously submitted processing requests (API3). Use comma (‘,’) to separate multiple request ids if retrieving status for more than 1 request.

Response	JSON object:
	{
	  "count": <number of records in status array>,
	  "status": [
	  <status>,
	  … …
	  ]
	}
	
	Each element in status array is a JSON object:
	{
	  "rid": <request id>,
	  "status": <status name>
	  "details": <status details, see below>
	}
	
	<status name> is one of the following **case sensitive** values:

Status Name	**Description**
PROGRESS	Request is still being processed
SUCCESS	Request is processed successfully
RETRY	Request has failed for some reason, but is being retried
FAILURE	Request has failed

	<status details> for PROGRESS:
	{
	“step”: <current step>,
	“total”: <expected total number of steps>
	}
	
	<status details> for SUCCESS:
	{
	“In”: <original video file>,
	“out”: <processed video file>
	}
	
	<status details> for RETRY and FAILURE include last error message. Currently the format is:
	{
	“exc_message”: <exception message, if any>,
	“exc_type”: <exception type, if any>
	}
	But please note the format may change if we decide to mask error information (or pass more information) to client applications.

API 5: Get Download URLs

Desc	Get download URLs for the specified request ids
Method + URI	GET {host}/download/rid GET {host}/download/rid1,rid2,...,rid
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	Clients can request download URLs for finished processing requests. Use comma (‘,’) to separate request ids if retrieving download URLs for multiple processing requests.

Response	JSON object:
	{
	“count”: <number of records in links array>,
	“links”: [
	<link>,
	… ...
	]
	}
	
	Each element in links array is a JSON object:
	{

	“rid”: <request id>,
	“name”: <name of the video>
	“size”: <size of the video>
	“duration”: <duration of the video>
	“input”: <link of the video>
	“urls”: [
	{
	“name”: <name of the downloadable item>
	“files”: <links of the files by extension> [
	{ <file type>: <URL to download the corresponding file>},
	{<file type>: <URL to download the corresponding file>}
	]
	}
	]
	}
	
	For example, if a processor outputs both bvh and fbx files, then the download link object will look like:
	{
	“rid”: “1234567890”,
	“urls”:[{
	“files”: [
	{“bvh”: “https://.../…”},
	{“fbx”: “https://.../…”}
	]
	}]
	}
	
	Please note that if the specified request has not finished yet or has failed, the response will not include any download urls, and the link object will look like:
	{
	“rid”: “1234567890”
	}

API 6: List All Video Processing requests by Status

Desc	List past and current request ids Note: failed jobs and old jobs may be removed by system after a predefined retention period
Method + URI	GET {host}/list GET {host}/list/status1,...,status
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>

<table>
<tr><td>Request</td><td>Client can request to get list of existing request ids of current user

Client can specify one or multiple status value(s) to retrieve only request ids with the same status value(s). For example, GET /list/PROGRESS will only return list of requests that are still being processed</td></tr>
<tr><td>Response</td><td>JSON object:
{
"count": <number of records in the rids array>,
"list": [
{
"rid": "1234567890-1234567890-1234",
"fileName":"",
"fileSize":0,
"fileDuration":0,
"status": "PROGRESS",
"ctime": <creation time>,
"mtime": <last modification time>
},
... ...
]
}

Each element in list is a JSON object with the following fields defined:

<table><tr><th>Field</th><th>Description</th></tr><tr><td>rid</td><td>Request/emoji id</td></tr><tr><td>fileName</td><td>Input video file name</td></tr><tr><td>fileSize</td><td>Input video file size in bytes</td></tr><tr><td>fileDuration</td><td>Input video duration in seconds</td></tr><tr><td>status</td><td>Current status (STARTING, PROGRESS, SUCCESS, FAILURE, RETRY)</td></tr><tr><td>ctime</td><td>Creation time (milliseconds since epoch)</td></tr><tr><td>mtime</td><td>Last modification time (milliseconds since epoch)</td></tr></table></td></tr>
</table>

API 7: Minutes Balance

Desc	Retrieves Minutes Balance for an user
Method + URI	GET {host}/account/minutesBalance
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	n/a

Response	JSON object:
	{
	"minutes":<value>
	}

API 8: Input Video Information

Desc	Get Video information such as resolution, fps etc.
Method + URI	POST {host}/videoInfo
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>

Request	POST body should include a JSON object:
	{
	"url": <upload url>
	}
	
	<upload_url> should match url returned from GET /upload request AND the video needs to be uploaded to that url in GCS before calling this API

Response	JSON object: {"width":1080,"height":1080,"fps":30,"duration":3,"codec":"h264","size":186615}

Custom Character APIs

Note: To make uploaded model(s) available to all animation jobs, please make sure x-useruid HTTP(S) header should have **not** been passed to the {host}/auth API to get the session for API 1 and API 2 below.

API 1: Model Upload Url

Desc	Retrieves signed urls to upload 3d model data(fbx format) and thumbnail(preferably png format)
Method + URI	GET {host}/character/getModelUploadUrl
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	Query parameters: <name>: base name of the files (without extension) (optional) <modelExt>: file extension of the model file. Example: fbx (optional) <thumbExt>: file extension of the thumb file. Example: jpg (optional) <resumable>: 0 or 1(default) returns resumable or regular signed url (optional)

Response	JSON object:
	{
	“modelUrl”: signed url
	“thumbUrl”: signed url
	}
	
	After retrieving the urls, actual model & thumbnail upload are required to that storage urls. If ’resumable’ option is set in the request, we need one POST and one subsequent PUT request for each signed url, otherwise a single PUT request will do the job per url.
	
	POST request to url:
	<x-goog-resumable>: start (set in the request header)
	<location>: resumable url (set in the response header by server)
	
	PUT request to resumable url location/url:
	attach raw bytes of the model or thumbnail file in the request body.

API 2: Store Model

Desc	Store the asset paths returned from getModelUploadUrl in database

Method + URI	POST {host}/character/storeModel
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	Body parameters: <modelUrl>: model url returned from API 1 (optional if <modelId> is provided) <modelName>: model name (optional) <thumbUrl>: thumbnail url returned from API 1 (optional) <modelId>: model id to update existing model info (name or thumb) (optional if <modelUrl> is provided) <createThumb>: 0 (default) or 1, indicate if the thumbnail of the model needs to be generated (optional)

Response	JSON object:
	{
	“modelId: Unique model id that can be passed to video process API
	}

API 3: List Models

Desc	List models based on specific query or without
Method + URI	GET {host}/character/listModels
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	Query parameters: <modelId>: existing model id (optional) <searchToken>: for example search by model name (optional)

Response	JSON object:
	[
	{
	“Id: Unique model id that can be passed to video process API
	“name”: name of the model
	“thumb”: url of the thumbnail if exist
	“rigId”: rigTemplate id with which this model is associated with
	“ctime”: creation timestamp
	“mtime”: modification timestamp
	}
	]

Experimental Webhook APIs

Event Payload

Headers:

HTTP POST payloads that are delivered to your webhook's configured URL endpoint will contain the following headers:

X-DeepMotion-Signature: <signature>

Note: Signature is your client ID. It is supposed to be verified by your event handling code.

Body:

The event body is a JSON object described below:

```
{
    "eventType": <event type>,
    "data" <event data>
}
```

The following table explains the currently supported event types and their data sub-attributes. Data sub-attributes is also a JSON object:

eventType	**Description**	**data**	**Note**

job.completed	A task is completed	{
		"taskId": <request ID>
		“status”: <success|failure>
		}
			taskId is the rid that returned by the POST {host}/process API

API 1: Create a webhook endpoint

Desc	Create a webhook endpoint
Method + URI	POST {host}/webhook_endpoints
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>

Request	JSON object:
	{
	"url": <endpoint URL>,
	"events": <array of events that would register with this endpoint>
	}

Response	JSON object:
	{
	"id": <endpoint ID>,
	"object": "webhook_endpoint",
	"url": <endpoint URL>,
	"events": <array of events that would register with this endpoint>
	}

API 2: Retrieve a webhook endpoint

Desc	Retrieve a webhook endpoint
Method + URI	GET {host}/webhook_endpoints/<endpoint ID>
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	

Response	JSON object:
	Response:
	{
	"id": <endpoint ID>,
	"object": "webhook_endpoint",
	"url": <endpoint URL>,
	"events": <array of events that would register with this endpoint>
	}

API 3: List webhook endpoints

Desc	List webhook endpoints
Method + URI	GET {host}/webhook_endpoints
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	
Response	JSON object: { "count": <number of endpoints>, "endpoints": [{ "id": <endpoint ID>, "object": "webhook_endpoint", "url": <endpoint URL>, "events": <array of events that would register with this endpoint> }, ...

	] }

API 4: Update a webhook endpoint

Desc	Update a webhook endpoint
Method + URI	POST {host}/webhook_endpoints/<endpoint ID>
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>

Request	JSON object:
	{
	"url": <endpoint URL>,
	"events": <array of events that would register with this endpoint>
	}

Response	JSON object:
	{
	"id": <endpoint ID>,
	"object": "webhook_endpoint",
	"url": <endpoint URL>,
	"events": <array of events that would register with this endpoint>
	}

API 5: Delete a webhook endpoint

Desc	Delete a webhook endpoint
Method + URI	DELETE {host}/webhook_endpoints/<endpoint ID>
Header(s)	cookie:dmsess=<cookie-value-returned-from-authentication-api>
Request	

Response	JSON object:
	{
	"id": <endpoint ID>,
	"object": "webhook_endpoint",
	"deleted": true
	}